

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Smartphone Gesture Learning

Paulo Ricardo Duarte Coelho da Silva



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Henrique Lopes Cardoso

June 17, 2013

Smartphone Gesture Learning

Paulo Ricardo Duarte Coelho da Silva

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Gabriel David

External Examiner: Doctor Paulo Novais

Supervisor: Doctor Henrique Lopes Cardoso

June 17, 2013

Abstract

Smartphone capabilities have been increasing in the last years, and many applications have been developed in order to take advantage of these capabilities. Smartphone users, knowing that there are lots of applications to ease many daily tasks, or simply to have some fun, typically have the smartphone close by. One of the capabilities that have been explored is the detection of the physical state of the smartphone, through inertial sensors embedded in the smartphone. The smartphone can detect its own orientation and even detect if it is being moved through sensors like the accelerometer, linear acceleration sensor or the gyroscope. Given that the smartphone can detect its own physical state and the fact that users often have their smartphone close by, the opportunity of developing a new natural and intuitive way of interacting with the smartphone arises, and this opportunity is related with gestures. Using the embedded accelerometer, linear acceleration sensor or gyroscope, the smartphone can detect if the user made some movement with the hand that is holding the smartphone. The objective of this Dissertation is to develop a software framework to be used in Android applications to render the applications capable of detecting gestures that the user makes with the hand that is holding the smartphone, sparing an Android developer the effort of implementing such functionality each time an Android application, which makes use of gestures as a means of interaction with the user, is to be developed. The idea is to have gestures embedded that allow an application incorporating this framework to recognize gestures right after the development phase, sparing the application user, or the developer, the effort of training the gestures. The gesture recognition capability is carried out by a Hidden Markov model approach, in a user independent setting, and it was achieved an average recognition accuracy of 97.8% using the gyroscope and the linear acceleration sensor on an alphabet of 8 gestures, and an average accuracy of 85.1% using the accelerometer and the gyroscope on an alphabet of 24 gestures. Smartphone gesture recognition has been used in several research areas, as health care, monitoring systems, or user commodity. One Android application using this framework could be used, for instance, to remotely control an electronic device, or trigger an action in the smartphone. Given the promising results that have been achieved, the next steps in terms of future work concern exploiting the developed framework in the development of a real application, taking advantage of this new interface for user interaction.

Resumo

As capacidades dos *smartphones* têm vindo a verificar um aumento nos últimos anos, e muitas aplicações têm sido desenvolvidas no sentido de tirar partido destas capacidades. Os utilizadores dos *smartphones*, sabendo que há várias aplicações para facilitar tarefas do dia-a-dia, ou simplesmente para se divertirem, tipicamente têm o seu *smartphone* por perto. Uma das capacidades que tem vindo a ser explorada é a capacidade de o *smartphone* detetar o seu estado físico, através de sensores inerciais incorporados no mesmo. O *smartphone* consegue detetar a sua própria orientação e até detetar se está a ser movido através de sensores como o acelerómetro, o sensor de aceleração linear ou o giroscópio. Uma vez que o *smartphone* consegue detetar o seu estado físico e o facto de que o utilizador o tem frequentemente por perto, surgiu a oportunidade de criar um novo modo de interação natural e intuitiva com o *smartphone*, e esta oportunidade é relacionada com os gestos. Através do acelerómetro, sensor de aceleração linear ou o giroscópio incorporados no *smartphone*, este pode detetar se o utilizador fez algum movimento com a mão que está a segurar o *smartphone*. O objetivo desta Dissertação é desenvolver uma *framework* para facilitar o desenvolvimento de aplicações para a plataforma Android, que faça com que estas aplicações sejam capazes de detetar gestos que o utilizador faça com a mão que segura o *smartphone*, poupando a quem está a desenvolver a aplicação Android o esforço de implementar tal funcionalidade de cada vez que uma aplicação Android que faça uso de gestos como um modo de interação com o utilizador seja implementada. A ideia é ter gestos incorporados na *framework* que permitem que uma aplicação que a incorpore possa ser usada tirando partido dos gestos mal a fase de desenvolvimento acaba, poupando ao utilizador da aplicação Android, ou a quem desenvolveu a aplicação o esforço de treinar os gestos. A capacidade de reconhecimento de gestos foi implementada usando *Hidden Markov Models* e foi atingida uma percentagem média de acerto de 97.8%, usando o giroscópio e o sensor de aceleração linear com um alfabeto de 8 gestos, e uma percentagem média de acerto de 85.1% usando o giroscópio e o acelerómetro num alfabeto de 24 gestos. O reconhecimento de gestos através de um *smartphone* tem vindo a ser explorado em várias áreas de investigação, nomeadamente saúde, sistemas de monitorização ou comodidade do utilizador. Uma aplicação Android que use esta *framework* pode ser usada, por exemplo, para controlar remotamente um aparelho eletrónico, ou despoletar uma ação no próprio *smartphone*. Tendo em conta que os resultados atingidos são promissores, os próximos passos em termos de trabalho futuro concernem a exploração da *framework* desenvolvida numa aplicação real, tirando partido desta nova interface destinada à interação com o utilizador.

Acknowledgements

First of all, I want to thank to Faculty of Engineering of the University of Porto and Fraunhofer, for allowing me to grow either academic and professionally during the current semester, and also to my supervisors, Henrique Lopes Cardoso and David Ribeiro. Obviously, without their efforts, any of this would have been possible. I also want to thank my friends, girlfriend and family for their support, even if it was not directly related to this. And finally, for those who for stronger reasons could not see me getting here, I want to give my special thanks to you. Cheers!

Paulo Silva

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation and Objectives	2
1.3	Main Results	2
1.4	Structure of the Dissertation	3
2	State of the Art Review	5
2.1	Related Work	5
2.1.1	Using Smartphones	5
2.1.2	Using Other Hand-Held Devices	6
2.1.3	MEMS Sensors Used in Other Areas	11
2.1.4	Tilt Calculation Using MEMS Accelerometers	12
2.2	Algorithms	13
2.2.1	Dynamic Time Warping	13
2.2.2	Hidden Markov Models	15
2.2.3	Support Vector Machines	19
2.3	Conclusions	21
3	Gesture Recognition Using a Smartphone	23
3.1	Rationale	23
3.2	Smartphone Sensors	24
3.3	Gesture Alphabet	25
3.4	Approach to Gesture Recognition	26
3.5	Conclusions	27
4	A Tool for Gesture Recognition	29
4.1	Functionalities Identified	29
4.2	Implementation of Each Functionality	30
4.3	Conclusions	34
5	Tests And Results	35
5.1	Setup	35
5.2	Results	38
5.2.1	Using the Accelerometer and the Gyroscope	39
5.2.2	Using the Linear Acceleration Sensor and the Gyroscope	40
5.2.3	Using only the Accelerometer	42
5.3	Result Analysis	44

CONTENTS

6	Conclusions and Future Work	49
6.1	Accomplishments	50
6.2	Future Work	50
	References	53
A	Sample Containing Test Results	55
A.1	Results Output	56
B	About the Framwork Usage	77
B.1	Gesture Sample File Format	77
B.2	Functionalities	78

List of Figures

2.1	Path to calculate the distance between two time series according to DTW	14
2.2	Correspondence established by DTW to corresponding points on two time series	15
3.1	Illustration of a translation and of a rotation	24
3.2	Illustration of the gesture alphabet used, found in [LWZ⁺09]	26
5.1	Illustration of the three orientations of the smartphone while collecting gesture samples	37
5.2	Illustration of the similarity between different gestures in the absence of gravity .	46
5.3	Illustration of the differences between different gestures in the presence of gravity	46

LIST OF FIGURES

List of Tables

5.1	Average recognition of the five datasets using all directions, using the accelerometer and the gyroscope.	39
5.2	Average recognition of the five datasets using the smartphone with it's screen up, using the accelerometer and the gyroscope.	39
5.3	Average recognition of the five datasets using the smartphone's with it's screen vertical, facing the user, using the accelerometer and the gyroscope.	40
5.4	Average recognition of the five datasets using the smartphone with it's screen horizontal, facing the user, using the accelerometer and the gyroscope.	40
5.5	Average recognition of the five datasets using all directions, using the linear acceleration sensor and the gyroscope.	41
5.6	Average recognition of the five datasets using the smartphone with its screen up, using the linear acceleration sensor and the gyroscope.	41
5.7	Average recognition of the five datasets using the smartphone's with it's screen vertical facing the user, using the linear acceleration sensor and the gyroscope. . .	41
5.8	Average recognition of the five datasets using the direction with the smartphone's screen horizontal facing the user, using the linear acceleration sensor and the gyroscope.	42
5.9	Average recognition of the five datasets using all directions, using only the accelerometer.	42
5.10	Average recognition of the five datasets using the smartphone's with it's screen up, using only the accelerometer.	43
5.11	Average recognition of the five datasets using the smartphone's with it's screen vertical, facing the user, using only the accelerometer.	43
5.12	Average recognition of the five datasets using the smartphone's with it's screen horizontal, facing the user, using only the accelerometer.	43

LIST OF TABLES

Abbreviations

MEMS	Microelectromechanical Systems
SVM	Support Vector Machine
HMM	Hidden Markov Models
Wiimote	Nintendo Wii Remote
TV	Television
VCR	Video Cassette Recorder
RFID	Radio-frequency Identification
LED	Light Emitting Diode
LTW	Linear Time Warping
DTW	Dynamic Time Warping
GPS	Global Positioning System
AP	Affinity Propagation
CS	Compressive Sensing
DCM	Direction Cosine Matrix
API	Applicational Programing Interface
DVD	Digital Versatile Disc
RBF	Radial Basis Function

Chapter 1

Introduction

In the last few years, smartphones have experienced a great improvement in their capabilities in terms of diversity and power. Many applications have been developed in order to aid their users in the daily tasks and hence, users have their smartphone around for most of the time. One of the capabilities that were included in smartphones is the capability to detect its own physical state due to some of the nowadays embedded sensors.

This capability, together with the great proximity of the smartphone to the user, has created an opportunity to explore gestures as a new and natural way of interaction with the smartphone, given that, in this context, a gesture is a 3D hand movement performed by the user while holding the smartphone with the hand, not allowing the smartphone to slide in the hand. The sensors most amenable to be used with this goal in mind are the accelerometer, capable of measuring the device's proper acceleration, and the gyroscope, capable of measuring the rotation imposed to the device.

1.1 Context

The objective of this Dissertation is to create a software framework to be used in Android applications, capable of recognizing and learning gestures, given that a gesture is a movement that a smartphone user makes with his hand, while holding the smartphone in the hand. An Android application that integrates this framework will be capable of recognizing some gestures right after its development, not requiring training effort from the application user in order to use it.

The state of the art regarding gestures as a way of interacting with the smartphone was studied in order to learn about other approaches to gestures as a way of interacting with smartphones, and relevant algorithms have been investigated. The aim was to identify challenges posed to other similar systems and how they were overcome, what algorithms were used and under what circumstances, and what is the performance of those systems and algorithms.

This Dissertation project is included in the study plan of the Integrated Master in Informatics and Computing Engineering at the Faculty of Engineering of the University of Porto, was proposed and developed at Fraunhofer Portugal.

1.2 Motivation and Objectives

The aim of the proposed software framework is to facilitate the development of Android applications which will make use of the sensors embedded in the smartphone to recognize gestures. The Android application will not detect gestures directly from the sensors embedded in the smartphone, but will instead exploit the gesture recognition facilities offered by the framework. The framework itself will interact with the sensors and will detect and process movements.

In order to allow a more fluid usage of the framework and of an Android application that integrates this framework, there will be gestures embedded on the framework, that will spare the Android application user the effort of embedding the gestures, and will spare the Android developer the effort of embedding the gestures, as Kela *et al.* state:

"would be nice if the system included a user-independent pre-trained library of typical navigation gestures enabling instant usage" ([KKM⁺05], p.296).

As the proposed software framework will be used on many kinds of Android applications, and as each application will be used by different users, it is important to achieve a user-independent gesture recognition, that is, it is important that gestures from the same set are recognized when performed by different people. User-independent gesture recognition is a challenge, already identified by other authors ([KKM⁺05, AV10, CHL13, HVL10, KARY12]).

The main objectives of this Dissertation are to develop the mentioned software framework, embed a pre-defined set of gestures in the software framework and enable the software framework to perform user-independent gesture recognition.

1.3 Main Results

As one of the goals is achieve user-independent recognition, the recognition of the gestures is carried by Hidden Markov Models (HMM) which is a statistical based strategy, used to identify patterns in time series (and initially used in the speech recognition field), that takes many samples of a given pattern in order to create a model of that pattern. Given that the patterns to be recognized are gestures, by providing samples from several people to create such models, user independent recognition was achieved, as suggested by Khan *et al.* [KARY12].

The samples from a gyroscope, a linear acceleration sensor and an accelerometer were used with the goal of creating one HMM per gesture. Due to the hardware capabilities of the smartphone used, a Samsung Galaxy Nexus, running the Android 4.0.1 platform, the readings from the mentioned sensors are reported at the same rate and have already noise filtering techniques implemented, so none of the existing state of the art noise filtering techniques was used.

The system was developed using an already made Java implementation of HMM called jahmm [Fra], under the New BSD License. This HMM implementation allows the creation of models with various data sequences, so once that the sensors used have the same sampling rate, and thus, streams from the sensors during the execution of a gesture have the same number of readings from the sensors, the data from the sensors was passed to the library, and multidimensional HMMs were created. For instance using an accelerometer and a gyroscope, each one reporting 3 dimensional values, 6 dimensional HMMs were created. Hidden Markov Models will be addressed in detail on Chapter 2.

It could be concluded that the usage of a gyroscope with an accelerometer, or a linear acceleration sensor provides better gesture recognition accuracies than the usage of only an accelerometer.

The proposed software framework can be used to help develop several kinds of Android applications, relative to many areas.

1.4 Structure of the Dissertation

The rest of this Dissertation is structured as follows. In Chapter 2, the state of the art regarding similar systems and relevant algorithms is presented. In Chapter 3, the proposed goals and usage of concepts found during the state of the art review is presented. Implementation details are presented in Chapter 4, and in Chapter 5 the tests made are explained and the results and conclusions about the results are exposed. In Chapter 6 all the main conclusions are drawn and the future work is presented.

Introduction

Chapter 2

State of the Art Review

In order to learn about the more recent developments related to the system to be implemented, the state of the art was reviewed. The aim was to learn about similar systems and to identify which problems were solved, which innovations were brought and which difficulties were overcome. The relevant algorithms related with similar systems were also reviewed in order to understand the advantages and disadvantages of each algorithm, and the circumstances that best suit each one.

Developing new testing methodologies is not the aim of this project, so the state-of-the art review did not focus this area. However other authors tested their approaches, and the tests to our approach will be performed based on tests performed by other authors.

2.1 Related Work

In the last few years, 3D gesture recognition has been an active research topic due to many reasons, being one of them the inclusion of Microelectromechanical System (MEMS) sensors in smartphones and the proximity between smartphones and their users. Some of these sensors, for instance, the accelerometer, the magnetometer, the gyroscope or even the camera, can be used to detect changes in the orientation and position of the smartphone. The focus of this state of the art review was the usage of the accelerometer and gyroscope. First, it is presented work directly related with hand gesture recognition using a smartphone embedded accelerometer, and then, using other kinds of hand-held accelerometer powered devices, like console remote controllers. After that, applications of inertial sensors in order to detect movements on areas other than gesture recognition are presented, and is finished with an introduction on how to use accelerometers to calculate tilt.

2.1.1 Using Smartphones

Prekopcsák [Pre08], presented a real time accelerometer-based hand gesture recognition system, with an automatic segmentation method, instead of the usual button based method, but only the ac-

celerometer data gathering was performed in a smartphone. All the segmentation and recognition process was implemented on a personal computer which received the data from the smartphone via a Bluetooth connection. This segmentation generated 3 dimensional vectors of variable finite size containing acceleration data. After the segmentation phase, two approaches were implemented to perform the gesture recognition: Hidden Markov Models (HMM) and Support Vector Machines (SVM). For the HMM approach, Prekopcsák used the Baum-Welsh algorithm for training. Prekopcsák did not make any transformation to the data from the accelerometer and refers to the work of Pylvaininen [Pyl05] where it is stated that no transformations to the data coming from the accelerometer are needed, once that the data in its raw format is well suited for the HMM.

For the SVM approach, the 3 dimensional vectors with acceleration data were transformed in 10 dimensional vectors with statistical data. The data contemplated was not clearly defined, but Prekopcsák refers that statistical data representing minimum, maximum, mean and length are used.

Prekopcsák affirmed that the accuracy of the recognizers is affected by the number of training samples used and finishes stating that for a set of 10 gestures, accuracy of 97.4% was achieved using HMM, and an accuracy of 96% was achieved using SVM.

Prekopcsák used an available Java library called jahmm[Fra] to implement the HMMs.

Khan *et al.* [KARY12] proposed a system called Gesthaar, that uses Haar transform and SVM for accelerometer-based gesture recognition. Gesthaar was tested using the uWave gesture library [LWZ⁺09], and a 99% recognition accuracy was achieved. Khan *et al.* affirmed that their system eliminates the necessity of personalized training and template adaptation, and to establish their system as user-independent, they showed that user-independence can be achieved during the training phase, providing samples from multiple subjects.

The Haar transform is a type of discrete wavelet transform, and is used to get features that can represent patterns of the gestures. These features are then used in a machine learning algorithm, in this case, SVM. In order to improve the SVM performance, training data was normalized.

Gesthaar managed to achieve a recognition accuracy of near 99% for every of the 8 gestures used. Khan *et al.* concluded that Gesthaar recognition accuracy is about the same in either user-dependent or user-independent training configuration. In a user-independent configuration the recognition accuracy drops slightly, but still remains above 99%, and also, they showed that the recognition accuracy does not drop consistently as more training samples from other users are included, and so they affirm that this method can be used to user-independent recognition.

2.1.2 Using Other Hand-Held Devices

Liu *et al.* [LWZ⁺09] presented the uWave, an accelerometer-based gesture recognition algorithm based on Dynamic Time Warping (DTW). uWave was evaluated with a 4000 gesture sample set, collected from 8 users, representing the same 8 gestures used by Kela *et al.* [KKM⁺05]. uWave also has a functionality that allows the adaptation of a gesture template to the user. With a single training sample for each gesture, uWave achieves an accuracy of 98.6% with gesture template adaptation and 93.5% without adaptation, for user-dependent recognition. Liu *et al.* state that

"HMM-based methods became the mainstream because they are more scalable toward a large vocabulary and can better benefit from a large set of training data. However, DTW is still very effective in coping with limited training data and a small vocabulary, which matches up well with personalized gesture-based interaction" ([LWZ⁺09], p. 2)

rendering the HMM an impracticable approach in an application that demands a minimal training effort.

The key components of uWave are acceleration quantization, DTW and gesture template adaptation. The gestures are treated like a time series of forces applied to a handheld device, so the aim is to match two time series of forces representing the gesture.

uWave quantizes the acceleration data before advancing to the gesture template matching. Quantization reduces the floating point computation converting the acceleration data into a discrete value, and reduces the length of input time series. Quantization also improves recognition rate once it reduces noise and minor hand tilt, as Liu *et al.* affirmed.

The implemented quantization method consists of two steps. The first one is compression, where the time series of acceleration are temporally compressed, and in the second step, the acceleration data is converted into one of 33 levels according to g value. The second step does not classify the data linearly, given that Liu *et al.* found that most of the movements have acceleration values between $-g$ and $+g$ and very few go beyond $+2g$ or $-2g$.

Liu *et al.* concluded that one person introduces some variations in the gesture over time, so template user adaptation is needed. To implement gesture template adaptation, uWave keeps two templates of each gesture generated in two different days. To match a gesture input with a template, uWave compares the input with both templates of each gesture, and the match is made between the input gesture and the template with smaller distance according to DTW. Two template adaptation strategies were implemented. One, which they called Positive Update consists of replacing the older template, if both templates of one gesture are more than one day old, and the gesture was successfully recognized. The other strategy, called Negative Update consists of replacing the older template if the gesture was not correctly recognized. Both of the strategies imply user feedback.

The recognition accuracy of uWave was tested in a user-dependent setting, once the objective of uWave is personalized gesture recognition. The accuracy achieved was of 97.4% for Positive Update, 98.6% for Negative Update and 93.5% without adaptation. Concerning user-dependent, or user-independent recognition, Liu *et al.* stated that user-independent gesture recognition is difficult, once that even the same person does not exactly repeat a gesture, and to improve the accuracy of user-independent recognition, a statistical method and a large set of training data from various subjects is needed.

Akl and Valaei [AV10] proposed a system to recognize accelerometer-based gestures that employs DTW and Affinity Propagation (AP) for training and Compressive Sensing (CS) for recognition of 18 gestures, in both user-dependent and independent settings. Their system was submitted to user-dependent and user-independent tests. For the user-dependent configuration, the input gestures are compared by DTW to the gesture templates obtained during the training phase.

For user-independent recognition, besides DTW, CS is used. For user-dependent recognition, the system achieves a recognition accuracy of 99.79% for the 18 gestures, and for user-independent recognition, the system achieves a recognition accuracy of 96.89% for a set of 8 gestures.

The first step of Akl and Valaee's approach, after gesture signal gathering, is temporal compression. With this, it is possible to filter signal variations not intrinsic to the gesture itself and to reduce the size of the acceleration signal, which leads to computational cost reduction.

Affinity propagation is a clustering algorithm that, as Akl and Valaee stated "simultaneously considers all data points as potential exemplars and recursively transmits real-valued messages until a good set of exemplars and clusters emerge." p. 2271.

To use CS, Akl and Valaee assume the gestures are "sparse since the hand follows a smooth trajectory while performing a gesture" p. 2271.

For the user-dependent training, a number of gesture repetitions were randomly chosen to create a training set. These training repetitions were temporally compressed, and then provided to DTW, and next to AP. For the user-independent training, a number of gesture repetitions from not more than 6 users were randomly selected to create a training set. The samples were temporally compressed and then, as for the user-dependent training, provided to DTW and next to AP.

For testing, either the user-dependent or independent setting used temporal compression and DTW after collecting accelerometer data, in order to compare the acquired signal to the ones gathered during training.

For user-dependent recognition, the test was performed with one user at a time, and the average accuracy for all users was computed, given that with 3 training repetitions, a recognition accuracy of 99.79% was achieved.

For the user-independent recognition, the test is performed with the unused gesture samples made by the users selected to compose the gesture training set added to the repetitions of the other users. After temporally compressed the gesture samples were compared, but in the user-independent setting, the gestures were compared by DTW and after that by CS. The tests were run with training gestures from 3 users. Akl and Valaee observed that the recognition accuracy dropped as the number of included gestures increased. Independently of that, the recognition accuracy was always greater for the gestures performed by users whose gestures were used during training. The average recognition accuracy for 8 gestures was about 97%, and for 10 gestures about 96%.

Cheema *et al.* [CHL13] studied and compared 4 machine learning algorithms and performed experiments to examine relationships between variables affecting recognition accuracy in a game environment, using a Nintendo Wiimote, the Nintendo Wii console wireless controller used to control the games via hand movements and button pressing while holding the controller in the hand. Cheema *et al.* also compared the impact of size of gesture set, amount of training data and choice of classifier in a user-dependent and a user-independent training configuration, and compared various classifiers - AdaBoost, SVM, Bayes Network and Decision Trees - with a linear classifier based on Rubine's Algorithm [Rub91].

In their work, Cheema *et al.* performed two experiments. In one of the experiments, using the gesture dataset composed by 25 gestures constructed by Hoffman *et al.* [HVL10] they gathered gesture data with the Wiimote and used the referred classifiers to recognize the gestures in a user-dependent and user-independent training configuration.

They concluded that in a user-dependent training configuration, the Bayes Network and the SVM classifiers have the greater average recognition accuracy. The recognition accuracy increased when the number of training samples increased and they achieved a recognition accuracy of 98.1% with the Bayes classifier, and 98.2% with the SVM classifier. However, for the user-independent training configuration, the best classifiers were SVM and AdaBoost. In a user-dependent configuration they were able to achieve an accuracy of 98.06% with both the SVM and AdaBoost classifiers.

The second experiment was to recognize the gestures in a video game environment, once the gestures may be involuntarily performed in a different way, as they affirm:

"In a hypothetical video game, players can be expected to perform gestures under stress induced by the game environment (e.g. fighting enemies to survive). It is likely that gestures performed in such situations may vary significantly from those collected in an environment which does not induce stress in players (e.g., a simple data gathering exercise)." ([CHL13], p. 17)

Cheema *et al.* concluded that the linear classifier is the algorithm, between the ones compared, that can achieve the best recognition accuracy but they suggest the usage of a two step algorithm with the linear classifier as a first step and in case of recognition fail, the Bayes or SVM classifier in a dependent or independent user configuration.

Kela *et al.* [KKM⁺05] studied accelerometer-based gesture control as a "supplementary or an alternative interaction modality" ([KKM⁺05], p. 285) and performed two user studies. One of the studies had the objective of finding gestures to control a design environment and some electronic devices (TV, VCR, and lights) and made Kela *et al.* conclude that different people prefer different gestures to produce the same effect. The other user study had the objective of evaluating the usefulness of the gesture modality compared to interaction modalities like speech, RFID, a laser-tracked pen and a PDA stylus. Kela *et al.* concluded that gestures are a natural modality interaction suited for some tasks, but for other tasks gestures are better used as a supplementary modality of interaction. Kela *et al.* also state that an application using gestures requires about 100% recognition accuracy to potentiate user satisfaction. Training the application should also be an easy task for the user. Many gesture recognition errors, or a laborious training may lead the users to feel uncomfortable and abandon gestures as an interaction method. Stating that the training effort must be simple, Kela *et al.* put away any recognition method that requires many training samples, or does not achieve a near 100% accuracy rate with minimal training effort. Also when discussing about their test results, Kela *et al.* stated:

"During the interviews, there were comments that it would be nice if the system included a user-independent pre-trained library of typical navigation gestures enabling instant usage and a separate training program for personalisation of the library" ([KKM⁺05], p. 296)

leaving suggestions about an architecture for an application to recognize and learn gestures using inertial sensor readings: the architecture must provide a near 100% recognition accuracy with a minimal training effort, and despite allowing the addition of user defined gestures, the application should include some predefined gestures rendering the application ready to use right after its development. The need for a user-dependent and user-independent recognition in the same application is implicitly referred here. User-dependence refers to the ability of identifying gestures from a user whose gestures are part of the training set, and user-independence refers to an ability of identifying gestures from a user whose gestures are not part of the training set.

In this work, Kela *et al.* used discrete HMM with Viterbi and Baum-Welsh algorithms to implement the gesture recognition process of 8 gestures, but before using the signal with the HMM's, the signal was preprocessed. This preprocessing consisted of normalizing all the gesture samples (because the gestures may vary in tempo and scale, various samples were collected) to the same length and amplitude, and then use a vector quantizer to reduce the dimensionality of the data to 1D sequences. These 1D sequences were finally used to train the HMM's in a user-dependent setting. 30 distinct samples for each gesture were collected from one person in two sessions over 2 days. To train the HMM's, 2 samples were used, and the remaining 28 were used to recognition, in order to calculate the recognition accuracy. In the second experiment, in order to reduce the number of training samples needed, Kela *et al.* added noise [Kay00] to the 2 samples used previously to train the HMM's, getting 4 samples to train each gesture. Training the HMM's with two training samples, Kela *et al.* achieved an accuracy of 87.2% but while using two original samples plus two noise-distorted samples, the recognition rate increased to 96.4%, given that when using only original training samples, a similar accuracy of 96.6% was achieved with 10 training samples. Kela *et al.* stated that HMM does not make good generalizations if the training set consists of only two samples.

Hoffman *et al.* [HVL10] in the referred work by Cheema *et al.* elaborated an experiment in order to conclude about how the number of gestures and the number of each gesture samples affect recognition accuracy using the linear and the AdaBoost classifier, in both user-dependent and independent setting. The device used to obtain gesture acceleration and angular velocity data was Nintendo Wii Remote and the Nintendo Wii MotionPlus.

Hoffman *et al.* performed two experiments using a user-dependent configuration: recognition accuracy using all the 25 gestures in the set and find the maximum recognition accuracy over as many gestures as possible. For the first experiment, a recognition accuracy of 98.5% was achieved using 15 training samples per gesture and using a linear classifier. For the second experiment, an accuracy of about 98% was achieved with 10 training samples per gesture, over 23 gestures using the linear classifier. The linear classifier had always outperformed the AdaBoost classifier in the user-dependent setting tests.

For the user-independent experiments, Hoffman *et al.* achieved, using the linear classifier, an accuracy recognition of 98.3%, recognizing 13 gestures with training data from 15 users and a total of 300 training samples. Again, the AdaBoost classifier was outperformed by the linear.

Hoffman *et al.* also concluded that for most of the cases, the usage of the Wii MotionPlus coupled with the Wiimote increased the recognition rate. In the other cases, the usage of the Wii MotionPlus did not lower the recognition accuracy.

Hoffman *et al.* concluded that the user-dependent recognizer provided higher recognition accuracy, and that the linear classifier provides more accurate results than the AdaBoost classifier. They affirmed that the natural expectation is the opposite, once the AdaBoost classifier is a more sophisticated technique, but the reason of this controversial is left unclear.

Wilson and Wilson [WW04] used a device called XWand and compared several gesture recognition algorithms: Linear Time Warping (LTW), Dynamic Time Warping (DTW) and Hidden Markov Models (HMM). XWand, is a handheld device with a wand shape, equipped with an accelerometer, a magnetometer, a gyroscope, a button, an FM transceiver, a flash-programmable microcontroller, an infra-red LED, a green and red LEDs and 4 AAA type batteries. This device is able to track movements performed by the holder's hand and send them wirelessly to a computer or other device capable of communicating via infra-red or FM communication. A set of seven gestures was used. Six subjects participated in their experiment providing training data, and performing 10 repetitions of each gesture acquiring a total of 420 training samples. The first three repetitions of each gesture were discarded due to learning effects, and the remaining seven were used for training. Although the experiment procedure is not very clear, the given values for recognition accuracy are 40.42% for LTW, 71.64% for DTW and 90.43% for HMM.

Wilson and Wilson concluded stating that LTW can not compensate for dynamic time warps, so DTW has advantage, but both of these algorithms can be affected by the order they receive training samples. Regarding HMM, the authors state that the fact of existing both left and right handed subjects performing the tests was a major factor in misclassification.

2.1.3 MEMS Sensors Used in Other Areas

Jonhson and Trivedi [JT11] used several sensors embedded in a smartphone attached to a car's windshield to recognize several movements performed by the car with the aim of classifying the driver's driving style. To do this, they implemented a DTW algorithm and used the smartphone camera, GPS, magnetometer, gyroscope and accelerometer signals. All the processing was done completely on the smartphone, an iPhone 4S. For the purpose of detection and classification of movement, the magnetometer, the accelerometer and the gyroscope signals were used in the DTW classifier. The data from the gyroscope and accelerometer was collected at a 25Hz rate to allow enough time for processing.

Taking into account that the device was attached to the car's windshield and to the dashboard, the device was subject of many vibrations inherent to the car. To compensate for this, a low-pass filter with a cut-off frequency of 1Hz was used. The maneuvers they were trying to detect were turn right, turn left, U-turn, hard right, hard left, hard U-Turn, swerve right and swerve left.

Jonhson and Trivedi concluded that the signal fusion implemented was useful and contributed to achieve greater recognition accuracy, having correctly identified 97% of the aggressive maneuvers. They also affirm that "DTW algorithm can accurately detect events with a very limited training set" ([JT11], p. 1614), meaning that DTW performs well with minimal training data, but only with a small gesture set.

Lou *et al.* [LXC⁺11] studied various error sources in MEMS sensors output and studied how to filter the noise from those sources. Using a gyroscope, an accelerometer and a magnetometer, Lou *et al.* tried to estimate the orientation, velocity, position and attitude of a mobile robot.

Lou *et al.* affirmed that every sensor has some faults. For instance, the gyroscope is not very sensitive to linear mechanical movements once its capability is to measure rotation but suffers from drift errors, and does not go back to zero value once the rotation stops. On the other hand, the accelerometer, capable of identify linear mechanical movements, suffers from noise caused with this type of movement. Lou *et al.* affirmed, in consequence to the exposed that

"we can obtain a relatively better estimation of the vehicle attitude by averaging data from the accelerometer and gyro instead of using the accelerometer or gyro data separately" ([LXC⁺11], p. 465)

Once the robot where the sensors were fixed moves, it caused the sensors to vibrate, and this affected the sensor readings. In order to compensate for this, Lou *et al.* used a moving average filtering algorithm, considering that the noise is randomly introduced in the sensor readings, and they were able to effectively reduce noise while maintaining the signal characteristic.

Lou *et al.* affirmed that fusing the signals from the various sensors is necessary in order to suppress the disadvantages of each sensor and achieve higher accuracy. A Direction Cosine Matrix (DCM) was used, in order to estimate the attitude and orientation of the robot. This was a 3×3 numerical matrix containing all the 9 degrees of freedom (DOF) estimated by the sensors, i.e. the 3D readings from the accelerometer, the gyroscope and the magnetometer. In order to deal with the noise in sensor readings, Lou *et al.* used a Kalman filter.

Herrera *et al.* [PKS⁺13] developed a fault detection and correction methodology for personal positioning systems for outdoor environments based on dead reckoning, in which they use a Kalman filter in order to deal with data fusion. In this approach, the Kalman filter is used to identify errors from other components of their system.

2.1.4 Tilt Calculation Using MEMS Accelerometers

Salhuana [Sal12] elaborated an application note about sensing tilt using accelerometers. Salhuana starts by clarifying that accelerometers are sensitive to both linear acceleration to the gravitational field, and that an accelerometer can only be used to sense rotation in the absence of linear acceleration and only if the rotation is not about the gravitational field vector.

After explaining how a MEMS accelerometer works, Salhuana introduces strategies used to calculate pitch and roll. It is exposed that the roll, pitch and yaw rotation matrices that transform a vector under the coordinate system of a smartphone by angles ϕ in roll, θ in pitch and ψ in yaw

have many orderings of application, and although they are equally valid, the final result depends on the ordering of application of these matrices, so it is mandatory to define in which order the matrices will be applied. Various ways of determining the values of the pitch and roll angles are then exposed.

Salhuana explains several calculations that can be made using an accelerometer output, including detection of rotations other than rotations about the gravitational field and detection of the tilt angle from the horizontal, but is very clear that an accelerometer can not be used to detect every rotation to which a device is subject to.

To test the recognition accuracy of their approaches, the authors reported that gestures from many subjects were gathered. The gathering of the gestures was made in various sessions over some days, or over some weeks, and each subject repeated the same gestures a few times. The authors gathered a lot of gesture samples and used them during the test phases accordingly to the user-dependent or user-independent character of their approach. For the user-dependent case, the approaches developed were trained with the gestures of one user at a time and tested with gestures from that same user. For the user-independent case, the training and testing were carried with gestures from several users at a time.

2.2 Algorithms

During the state of the art review, the most relevant algorithms related to gesture learning and recognition were Dynamic Time Warping (DTW), Support Vector Machines (SVM) and Hidden Markov Models (HMM), for which they will be briefly explained here.

2.2.1 Dynamic Time Warping

DTW calculates, given a certain cost function, the matching cost between two time series. In the uWave [LWZ⁺09] case, the matching cost corresponds to the Euclidian distance between two points in the Euclidian space. Let $S[1..M]$ and $T[1..N]$ be the time series to be compared by DTW, given that the more similar S and T are, the lower is the value for the distance between these two time series returned by DTW. This algorithm, using dynamic programming, will establish a relation between the points of the two time series, taking into account that the two time series to be compared may not have the same size or amplitude. Hence, the goal is to identify which points of T correspond to each of the points in S , which due to the differences in scale and tempo on the two time series, will probably have different index numbers, and measure the distance between the two time series according to those correspondences.

Figure 2.1 illustrates the path found by DTW between two time series, using dynamic programming, which is used to calculate the distance between the two time series. The algorithm iterates on every point of each time series, beginning at the point (1,1) and ending at point (M, N). The optimal path between the beginning point and any intermediate point (i, j) is measured taking into account the distance between (1, 1), (i, j) and the previous neighbours of (i, j): (i-1, j), (i, j-1) and (i-1, j-1). The distance between (1, 1) and (i, j) is then the actual distance between the

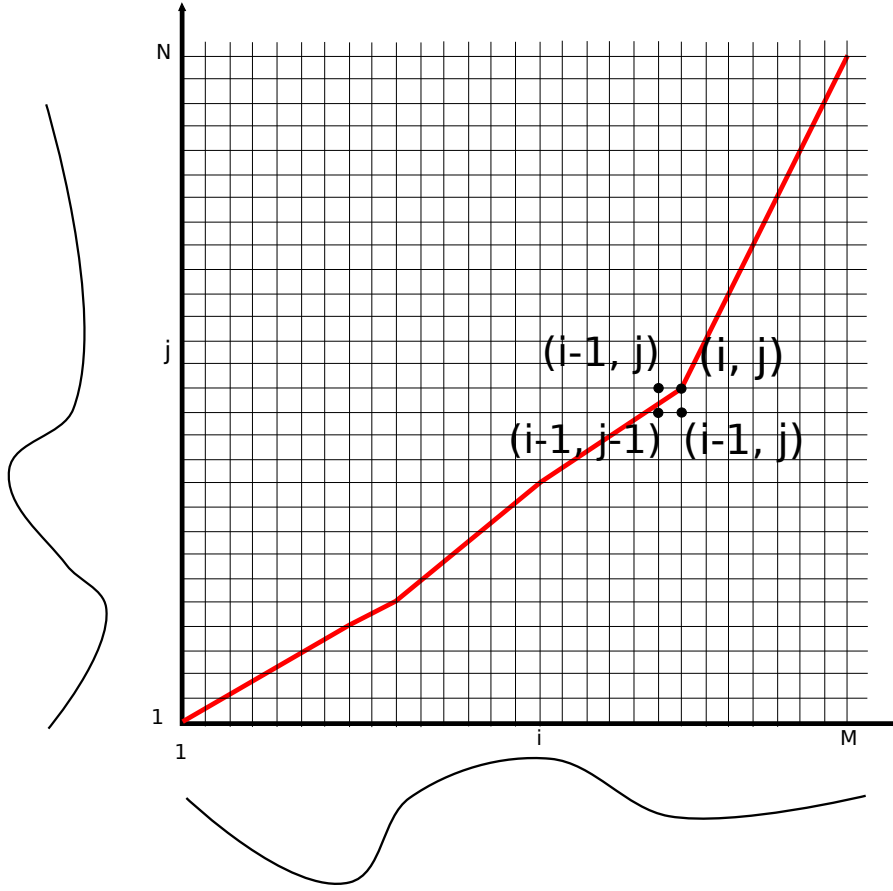


Figure 2.1: Path to calculate the distance between two time series according to DTW

referred points, plus the smallest distance computed between $(1, 1)$ and the referred neighbours of (i, j) . After visiting all the combination of points between the two time series, DTW returns the matching cost of the two time series as the sum of the matching costs between every pair of corresponding points.

Figure 2.2 illustrates some of the correspondences between points of two time series established by DTW. It can be seen that DTW does not match points because they have the same index in the time series, but tries to overcome the differences between the scale and amplitude of the time series, and establishes relations between the potential corresponding points. It can be observed, for instance, that DTW associates the local maximum or minimum of each time series. In the context of our work, the differences between two time series can be due, between other reasons, to the fact that one gesture is never made in the same way, even if it is made by the same person. The gesture can be made in more or less time, which makes the time series longer or shorter, or can be made with wider movements, which will affect the amplitude of the time series representing the gesture.

The DTW algorithm pseudo code can be written as follows:

```

double DTWdistance(timeSeries1: array[m], timeseries2: array[n]){
    DTW: array[m][n];
    for i:=0 to m
        DTW[i][0]:=infinity;
    for j:=0 to n
        DTW[0][j]:=infinity;
    for i:=0 to m
        for j:=0 to n
            distance:=d(s[i], t[j]);
            DTW[i][j]:=distance+min(DTW[i-1][j-1], DTW[i][j-1], DTW[i-1][j]);
    return DTW[m][n];
}

```

2.2.2 Hidden Markov Models

HMM is a statistical method, including several algorithms, each one with the goal of performing a certain task. Dugad and Desai [DD96] made a good compilation of the previous work of Rabiner [Rab90], where the HMMs are introduced as a speech recognition technique.

Dugad and Desai start by defining the concept of observation sequence:

"Suppose a person has three coins and is sitting inside a room tossing them in some sequence- this room is closed and what you are shown (on a display outside the room) is only the outcomes of his tossing TTHTHHTT. . . this will be called the observation sequence." ([DD96], p.1)

The coin tossing example is used to introduce the first concepts definition. Supposing that the observer is in the already referred conditions, he does not know in which sequence the three coins are being tossed, nor knows the bias of each coin. Next, if it is given that the third coin is biased to produce heads and all the coins have the same probability of being tossed, the observer will naturally expect a greater number of heads than tails in the final observation sequence. On the other hand, if the observer is given that besides the bias of the third coin (state), the probability of tossing the third coin knowing that the last tossed coin was the first or the second is equal to zero,

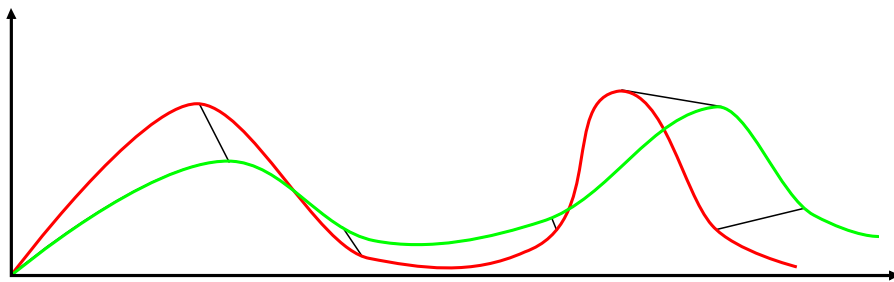


Figure 2.2: Correspondence established by DTW to corresponding points on two time series

and assuming that the first coin to be tossed was the first or the second, the number of heads and tails on the final sequence will not be very different in spite of the third's coin bias.

Given this example, Dugad and Desai conclude stating that the observation sequence depends on the individual bias, the transition probabilities between the various states (which coin is tossed) and which is the first state to begin with the observations. These three sets (individual bias of each coin, the transition probabilities from a coin to the next, and the set of probabilities of choosing the first state) characterize the HMM for this coin tossing experiment.

In order to provide a more general example, with more observation symbols than the two on the previous example (heads and tails), Dugad and Desai used an example in which they considered a set of N urns. Each urn contains a number of marbles of M distinct colors, given that inside each urn there are no two marbles of the same color. The experiment on this example consists of drawing marbles from the urns, being only shown the sequence of the drawn marbles.

To better understand the concepts related to HMMs, some notations must be defined (the same notation can be found on the papers of Dugad and Desai ([DD96], p. 2) and of Rabiner [Rab90]):

- N = number of states (urns) in the model
- M = total number of distinct observation symbols (marbles of M distinct colors)
- T = length of observation sequence i.e. the number of symbols observed
- $1, 2, \dots, N$ will denote the N urns respectively
- i_t denotes the state in which we are at time t
- $V = \{v_1, \dots, v_M\}$ the discrete set of possible observation symbols
- $\pi = \{\pi_i\}$, $\pi_i = P(i_1=i)$, the probability of being in state i at the beginning of the experiment i.e. at $t=1$
- $A = \{a_{ij}\}$ where $a_{ij} = P(i_{t+1}=j \mid i_t=i)$, the probability of being in state j at time $t+1$ given that we were in state i at time t We assume that a_{ij} 's are independent of time
- $B = \{b_j(k)\}$, $b_j(k) = P(v_k \text{ at } t \mid i_t=j)$ the probability of observing the symbol v_k given that we are in state j
- O_t will denote the observation symbol observed at instant t
- $\lambda = (A, B, \pi)$ will be used as a compact notation to denote an HMM

Dugad and Desai finished their key concept introduction explaining that given this model, an observation sequence is generated by choosing one of the urns, according to the the initial probability distribution π , to drawn the first marble, at time $t=1$, according to the probability distribution B (the distribution that specifies the probability of observation of a symbol v_k , from V , given a certain state). Then, according to the state transition probability A , the next urn is chosen

and a new marble is drawn. The process of drawing the second marble is repeated until a desired observation sequence of length T is reached.

Next, Dugad and Desai expose the three problems of HMMs:

1. Given the model $\lambda = (A, B, \pi)$, how to compute $P(O | \lambda)$, the probability of occurrence of an observation sequence $O=O_1, O_2, \dots, O_T$.
2. Given the model $\lambda = (A, B, \pi)$, how to choose a state sequence $I=i_1, i_2, \dots, i_T$ so that the joint probability of the observation sequence and the state sequence, $P(O, I | \lambda)$, is maximized.
3. How to adjust the HMM parameters, so that the previous probabilities, $P(O | \lambda)$ and $P(O, I | \lambda)$ are maximized.

The algorithms to solve the three problems are exposed.

To solve the first problem, the Forward-Backward algorithm is explained. Let $\alpha_t(i)$ be a forward variable defined as:

$$\alpha_t(i) = P(O_1, O_2, \dots, O_t, i_t = i | \lambda) \quad (2.1)$$

This variable represents the probability of the partial observation up to time t , and the occurrence of state i at the same time, given the model λ . Using this forward variable, Dugad and Desai conclude that:

1.

$$\alpha_1(i) = \pi_i b_i(O_1), 1 \leq i \leq N \quad (2.2)$$

2. for $t=1, 2, \dots, T-1, 1 \leq j \leq N$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(O_{t+1}) \quad (2.3)$$

3. and then:

$$P(O | \lambda) = \sum_{i=1}^N \alpha_T(i) \quad (2.4)$$

So, the desired probability can be calculated with the expression 2.4.

After that, let $\beta_t(i)$ be a backward variable defined as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \dots, O_T | i_t = i, \lambda) \quad (2.5)$$

$\beta_t(i)$ represents the probability of observation sequence from time $t+1$ to time T , given the model λ the state i at time t , which was not given on the forward variable. This will be useful to combine the forward and backward variables.

For the backward variable, it can be concluded that:

1.

$$\beta_T(i) = 1, 1 \leq i \leq N \quad (2.6)$$

2. for $t=T-1, T-2, \dots, 1, 1 \leq i \leq N$

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \quad (2.7)$$

3. and then:

$$P(O | \lambda) = \sum_{i=1}^N \pi_i b_i(O_1) \beta_1(i) \quad (2.8)$$

According to the backward variable, $P(O | \lambda)$ can be calculated as expressed in 2.8.

About the solution to the second problem, the maximization of $P(O, I | \lambda)$, by choosing a state sequence $I=i_1, i_2, \dots, i_T$, it is referred an algorithm called Viterbi Algorithm.

As Dugad and Desai state about the Viterbi Algorithm:

"is an inductive algorithm in which at each instant you keep the best (i.e. the one giving maximum probability) possible state sequence for each of the N states as the intermediate state for the desired observation sequence $O=O_1, O_2, \dots, O_T$." ([DD96], p.5)

For the third problem, a problem regarding the definition (or training) of the models, two algorithms are presented. The aim of these algorithms is to encode the observation sequences in the models, so that a model will identify other observation sequences with characteristics similar to the characteristics of the observation sequences encoded in the models.

One of the algorithms used is the Segmental K-means algorithm. This algorithm is used to adjust the parameters of the models in order to maximize $P(O, I | \lambda)$, where I is the sequence given by the Viterbi Algorithm. For this algorithm

1. a number ω of observation sequences is needed to train the model. N observation symbols are randomly chosen, and each symbol on each of the training sequences is associated with one of these random symbols from which its Euclidian distance is minimum, forming N clusters called states;
2. the initial probabilities and state transition probabilities are calculated;
3. the mean vector and covariance matrix for each state is calculated;
4. the distribution of the probability of observing each symbol on each state is computed;

5. it is found the optimal state sequence I according to the Viterbi Algorithm for each training sequence using the parameters found within steps 2 to 4, and if an observation symbol vector has an optimum state different from its initial state, this observation symbol vector is reassigned to a new state;
6. if any vector was reassigned, all steps from step 2 to this are repeated. Otherwise, the algorithm stops.

The other algorithm that contributes to the solution of the third HMM problem is the Baum-Welsh re-estimation algorithm. This algorithm adjusts the parameters of an already parametrized model, finding local maximums, so it has to receive models with reasonable parameters. Models parametrized by the Segmental K-means algorithm, for instance, are a good starting point to the Baum-Welsh re-estimation algorithm.

For more details about the algorithms exposed, please refer to [DD96] or [Rab90]. The work of Visser [Vis11] is also a good clarification of the key concepts of HMMs.

2.2.3 Support Vector Machines

The work of Hsu *et al.* [HCL03], which is a practical guide meant to introduce the SVM approach to starters who are not familiar with such approach, will serve as a guide to expose the key concepts regarding SVM.

Hsu *et al.* start by clarifying that SVM is a technique for data classification, and that usually, a classification technique requires a training and a testing data set. For usage with SVM, each instance of the training set contains what they call a "target value", that is, a label indicating the class of that instance, and several "attributes", also called features, or observed variables. Taking into account the training data, the SVM approach generates a model that predicts the target values of the test data, given the attributes of that same data. In order to accomplish this prediction, the SVM require the solution of the following optimization problem, that is applied to a set of instance-label pairs (x_i, y_i) :

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^L \xi_i \\ \text{subject to} \quad & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \end{aligned}$$

The purpose of the function ϕ is to map the training vectors x_i into a higher dimensional space, where SVM finds a linear separating hyperplane with the maximal margin. Hsu *et al.* clarify that C is a penalty parameter associated with errors and introduce the kernel function $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$. Hsu *et al.* present the four basic SVM kernel functions:

- linear: $K(x_i, x_j) = x_i^T x_j$.

- polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$, $\gamma > 0$.
- radial basis function (RBF): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$, $\gamma > 0$.
- sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$.

In the expressions exposed, γ , r , and d are the kernel parameters.

Next to that, Hsu *et al.* ([HCL03], p. 3) propose a procedure to use SVM:

- Transform data to the format of an SVM package
- Conduct simple scaling on the data
- Consider the RBF kernel $K(x, y) = e^{-\lambda \|x-y\|^2}$
- Use cross-validation to find the best parameter C and γ
- Use the best parameter C and γ to train the whole training set
- Test

Regarding data formats, it is required that each data instance is represented by a vector of real numbers, and if there are categorical attributes, it is recommended to convert them into vectors of numbers as well, but this time using binary data. If there are m categories, m numbers should be used and only one should be 1 and all the others 0. To better understand this data representation, Hsu *et al.* used an example where the attribute color has three categories defined as red, green, blue and each is represented, as (0,0,1), (0,1,0) and (1,0,0), respectively. Data scaling is also important before the data is applied to SVM.

The next problem to solve according to Hsu *et al.*, is the model selection. Although only four different kernel functions are presented in their work (the most basic), there are other functions, and is necessary to choose one, and after that, to find the values of the parameters C and γ . It is clarified that usually, the RBF kernel is a reasonable first choice, this statement is justified with three reasons. The first reason is the non-linearity property of this kernel function. The training samples are non-linearly mapped into a higher dimensional space, the RBF kernel can handle the case where there are classes and attributes which relate in a non-linear way. The second reason is the simplicity of the RBF kernel in terms of hyperparameters. The third reason is the numeric simplicity. The values of K_{ij} follow the rule $0 < K_{ij} \leq 1$, while, for instance the polynomial kernel may go to infinity, or to zero. About the sigmoid kernel, Hsu *et al.* state that it is not valid under certain circumstances.

In order to find the best values for the kernel parameters regarding a given problem, which are unknown beforehand, some search on those parameters is necessary. Hsu *et al.* recall on the fact that it may not be good to achieve high training accuracy once it can lead the model to be overfitted, and state that a cross validation is a good approach to obtain good model parameters while reducing the risk of causing the model to get overfitted. The process of cross-validation is introduced, and is clarified that it consists of dividing the training set into v subsets of equal size.

The classifier is trained using one subset, and tested using the remaining $v-1$ subsets. In order to find the best values for the kernel parameters, Hsu *et al.* recommend the usage of a "grid-search" where both parameters are assigned different values, and the combination that allows the final model to have the best results is chosen. Hsu *et al.* state that they found that "trying exponentially growing sequences of C and γ is a practical method to identify good parameters" ([HCL03], p. 5). It is also recommended that a coarse grid should be used first just to get an estimation of where the best values for the kernel parameters are, and then a finer search should be performed on the first region found. This might save time and computational burden.

Hsu *et al.* conclude stating that there are situations where the proposed approach may not be enough to achieve good results, but those situations are beyond the scope of their work ([HCL03]), which is a practical guide to introduce the key concepts of SVM to beginners, and that the proposed approach works well with data with not many attributes. The exact number of attributes that is suitable for their approach is left unclear, but it is stated that "If there are thousands of attributes, there may be a need to choose a subset of them before giving the data to SVM." [HCL03], p. 8)

2.3 Conclusions

During the state of the art review, several similar systems and implementation approaches came to be found. The most relevant algorithms for gesture learning and recognition were Hidden Markov Models (HMM), Dynamic Time Warping (DTW) and Support Vector Machines (SVM).

HMM is a statistical approach used in [WW04, KKM⁺05, Pre08]. Although this algorithm was first developed to be implemented in speech recognition, it proved to be quite useful and to be capable of accurate gesture recognition if dealing with a greater number of gestures, but as a statistical method, HMM requires an extensive training set. DTW, used in [WW04, LWZ⁺09, JT11], on the other hand, is an algorithm that tries to find patterns in two sequences, and is capable of dealing with sequences in different phases or amplitude. In this concrete case, DTW would be capable of recognizing two gesture sequences, even if one of the gestures was made at a higher speed, or started later than the other. DTW, in opposition to HMM, achieves a higher recognition rate if the number of gestures to identify is small, but requires minimal training effort, having achieved good results with one simple training example [LWZ⁺09]. SVM [Pre08, KARY12] is a supervised machine learning algorithm that can handle a bigger number of gestures, but needs a big training set and as [Pre08] states, HMM performs better than SVM.

Almost all the referred authors referred some kind of operation performed at signal level, before the signal was passed to recognition or learning process.

Although Prekopcsák [Pre08], as Pylvänäinen [Pyl05] suggests, did not use any noise filtering technique once that the accelerometer signal is suited for HMM in its raw format, Kela *et al.* [KKM⁺05], using also HMM, normalized the gesture samples to the same length and amplitude and converted the data into 1D sequences. Liu *et al.* [LWZ⁺09] used quantization in order to reduce the length of input time series and to reduce signal noise. Akl and Valaee [AV10], using DTW used temporal compression in order to reduce the size of acceleration data and filter out any

variation not intrinsic to the gesture itself. Jonhson and Trivedi [JT11], used a low-pass filter in order to compensate for the high vibration captured from the vehicle chassis. Khan *et al.* [KARY12] used a Haar transform in order to prepare the signal for the SVM algorithm implemented.

Wilson and Wilson [WW04] and Jonhson and Trivedi [JT11] used both gyroscope and accelerometer in order to estimate the orientation of a device and used it to recognize gestures. In order to deal with the readings from many sensors, they prepared their classifiers to receive all the signals, and instead of comparing the readings from one sensor they compared the readings from two sensors.

Lou *et al.* [LXC⁺11], using the Direction Cosine Matrix (DCM), grouped the readings from 3 sensors. Once that these were 3D readings, the DCM had 3 rows and 3 columns, and the Kalman filter they used operated in this matrix. Herrera *et al.* [PKS⁺13] used the Kalman filter in order to fuse the signals and detect error in the data from previous parts of their system.

Regarding user adaptation, Liu *et al.* [LWZ⁺09] implemented two gesture template update algorithms. Keeping two gesture templates per gesture, if one gesture is successfully recognized, the older template is replaced, or, in case of unsuccessful recognition, the gesture template of the right gesture is updated. In one way or another, the user must provide feedback about the recognition success. Khan *et al.* [KARY12], on the other hand, affirm that user-independence can be achieved including gestures from many subjects in the training phase, and this eliminates the need of adaptation. Also, developing an application and distribute it with the gestures embedded, makes the usage of the application more comfortable to the user [KKM⁺05], as the user does not have to insert the gestures in the application.

Given the state of the art review it is proposed to contribute with an integration of various components needed for gesture recognition with a smartphone's embedded inertial sensors, and to compile them in a software framework enabling a developer to rapidly incorporate a set of trained gestures in an Android application. The auxiliary tool needed will also be developed: a tool to generate the models and a tool to test the performance of the models generated. This will be exposed in more detail in Chapter 3.

Chapter 3

Gesture Recognition Using a Smartphone

In this Chapter, we expose the proposed goals and introduce how the concepts found during the state of the art review, studied in Chapter 2, were used.

3.1 Rationale

The main goal of this Dissertation, was to implement a software framework to facilitate the development of Android applications that make use of 3D gestures, given that a gesture is a movement that the smartphone user performs with the hand, while holding the smartphone with the moving hand, not allowing the smartphone to drift in the hand. This software framework is meant to be included in an Android application during the development time, and its function is to receive the sensors' readings during the execution of the application, and notify when a gesture is detected. From the developer point of view, it is only necessary to initialize the framework (details will be addressed in Chapter 4) and to specify what the Android application is supposed to do when the framework reports a gesture. The direct interaction with the sensors and gesture identification is not carried out by the developer, but by the framework. In order to accomplish this, various issues must be addressed.

The most important issue addressed is the recognition of the gestures in an user-independent setting. If an Android application is built using this framework, it will be distributed with the gesture models already embedded. A gesture model is the set of characteristics used to compare the time series composed of readings that the smartphone's inertial sensors report while the application user is performing a gesture. As the gesture models are already defined, enabling the user to use the application without having to add the gesture models himself, they must be generic enough to be able to deal with differences in the gesture samples collected from the sensors, but must not be excessively generic, in order to distinguish different gestures. It is known that different people

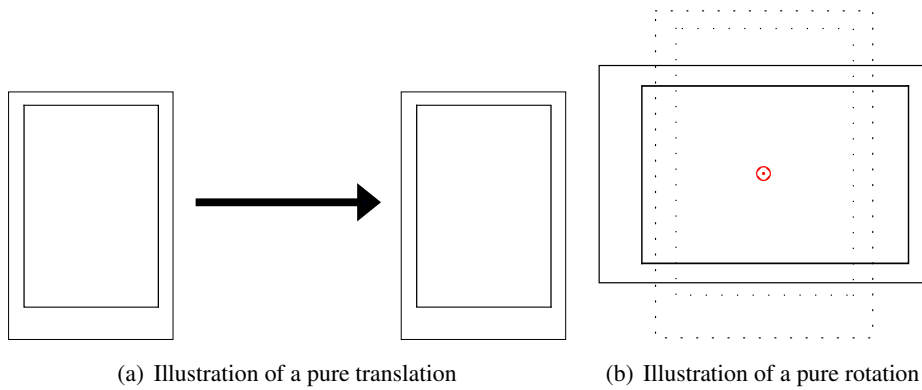


Figure 3.1: Illustration of a translation and of a rotation

will make the gestures in a different way, and that even the same person will not repeat exactly the same gesture twice. These two facts demand that the gesture recognition must be performed in a user-independent setting.

3.2 Smartphone Sensors

In order to try to improve the recognition accuracy, two sensors will be used: the gyroscope and the accelerometer. The accelerometer is a sensor capable of measuring the acceleration that a device is suffering. Because it is an inertial sensor, an accelerometer is able to sense the acceleration due to gravity plus the acceleration due to any movements of the smartphone in relation to the earth ([Sal12]). The gyroscope, also an inertial sensor, is capable of measuring the angular velocity on each one of the smartphone's 3D axes. The aim of using these two sensors is to include in the gesture models information about any translation and any rotation applied to the smartphone. In order to define what is a translation and a rotation, let's think of the smartphone as being only the particle that represents its center of mass. If the particle moves from one location to other, and during the movement keeps its orientation, for instance north, the movement can be described by the change of the coordinates of the particle in a 3D Euclidian referential and is called a pure translation (see Figure 3.1(a)). On the other hand, if the particle remains in the same coordinates of the referential, meaning that there was no translation, but its orientation changed, say from north to south, the particle suffered a rotation (see Figure 3.1(b)). To be noted that even if the particle moves in a circular trajectory, but its orientation remains the same, the particle is suffering a translation, but if the particle's movement describes the same circular trajectory and its orientation is, for instance, always in the direction of the center of the trajectory, the particle is suffering a translation and a rotation. Let's think of the earth's movement around the sun. Its rotation provokes day and night, because of the variation on the planet orientation, and its translation provokes the different seasons of the year because of the elliptical translation around the sun.

The usage of the gyroscope is then meant to more accurately detect rotations, given that the accelerometer can not detect a rotation in a plan perpendicular to the earth's gravity direction, and

a rotation can only be detected in the absence of linear acceleration ([Sal12]).

During the state of the art review, references to sensor fusion with the goal of detecting rotation were found ([LXC⁺11, PKS⁺13, JT11]) but such techniques were not used due to the hardware in the smartphone used, and to avoid computation during the recognition phase (more details will be explained in Chapter 4).

The smartphone available to help the development of this work makes available many sensors capable of detecting the device's orientation and movement. These sensors are listed below.

- Gyroscope - detects angular velocity
- Accelerometer - detects acceleration due to gravity and to movements of the smartphone relative to the earth.
- Magnetic Field - measures the strength of magnetic fields
- Orientation - measures roll, pitch and azimuth
- Rotation Vector - measures the rotation of the smartphone on each of the 3D axes relative to the earth
- Linear Acceleration - measures only acceleration due to movements of the smartphone - acceleration due to gravity is filtered out
- Gravity - measures only the acceleration due to gravity

The mentioned sensors report 3 dimensional values regarding each one of the 3 dimensional axes. The axes specification used in Android powered devices can be consulted in [Andb].

The sensors more interesting regarding the scope of this work are a gyroscope, an accelerometer and a linear acceleration sensor. More details can be found on Chapter 5.

3.3 Gesture Alphabet

The gesture alphabet used is a gesture identified by a Nokia research, found during the state of the art review, and used in the works of Kela ([KKM⁺05]) and Liu ([LWZ⁺09]), since it was not part of our work to investigate which are the most intuitive gestures that one makes while holding the smartphone with the hand, nor how to identify the currently more used gestures.

The gesture alphabet is illustrated in Figure 3.2 and as can be seen, is composed of 8 gestures of different natures and complexities. The dot in each gesture denotes the beginning of the gesture, and the arrow denotes the end of the gestures. The line connecting the dot and the arrow explains the trajectory of the gesture since the beginning to the end.

1	2	3	4
			
5	6	7	8
			

Figure 3.2: Illustration of the gesture alphabet used, found in [LWZ⁺09]

3.4 Approach to Gesture Recognition

As suggested by the state of the art review (see Chapter 2), the strategy that best suits an user-independent gesture recognition is Hidden Markov Models (HMM). The Support Vector Machines (SVM) also appears as a promising algorithm, but as HMM is suggested to perform best was used in the first place. Due to time restrictions, the SVM approach was not used. Algorithms using statistical approaches achieve higher recognition accuracy for a bigger gesture alphabet¹, but also need a very large training set, given that this large training set can be used to achieve user-independence, using gesture samples from many subjects, as stated in Chapter 5. This user-independence means that a model is flexible to allow some modifications in the gestures while maintaining the recognition accurate. This flexibility allows the user of an application that incorporates the framework to make each repetition of the same gesture with slight differences, and also, because of the modifications allowed, users that did not take part in the training set building can also have their gestures recognized.

In order to have the movements that a person holding a smartphone performs recognized as gestures, several samples of each gesture in the selected alphabet were collected from many people. The samples of each gesture were used to train the HMMs, given that one HMM is meant to identify only one gesture, so it was trained one HMM per gesture. Each HMM was trained using the Segmental K-means algorithm to obtain a good first estimation of the model, and then the HMMs were optimized using the Baum-Welsh re-estimation algorithm. According to the notation found on Section 2.2.2, the set of possible observation symbols V is the multidimensional set of possible combinations of values that can be returned by the sensors. Each sensor reports 3 dimensional values and more than one sensor will be used, so the dimensionality of each observation symbol is the number of sensors multiplied by 3. The number of possible observation symbols, M , depends on each sensor's range (the maximum and minimum value that the sensor reports on each axis) and resolution (the minimum difference between two different reported values). According

¹On the literature reviewed, the set of gestures that are meant to be recognized is called alphabet.

to its characteristics, subsets of observation symbols may be grouped into what is called of states, originating the set B , that is, the probability of observing the symbol v_k given the current state. In the current context, a state is a set of readings that have some common characteristics and, for instance, could be the set of readings that are obtained from the sensors when the smartphone is starting to move from left to right.

The recognition phase, when it is needed to classify a sample, is carried by the Viterbi algorithm. The Viterbi algorithm evaluates the probability of a given sample has been generated by a certain HMM, taking also the most probable state sequence within the HMM into account. Given that the alphabet is composed of more than one gesture, the Viterbi algorithm is used with each HMM and computes, for each HMM, the probability of generation of the given sequence. The probability values associated with each model are saved and compared. The sample is classified after the HMM that was attributed the higher probability of generation of the given sample, by the Viterbi algorithm.

Given that it may happen that a sample does not represent one of the gestures in the alphabet, it is necessary to evaluate if the results given by the Viterbi algorithm are reliable. It may happen that a sample not representing any gesture has an associated probability value of zero, meaning that the Viterbi algorithm do not assign any probability of generation of that sample to any trained HMM, but it may also happen that some HMM is assigned a probability greater than zero to a sample regarding a movement that does not correspond to any gesture in the alphabet. However, even if a gesture is misclassified in this circumstance, the HMM which is supposed to be associated to that sample, is assigned by the Viterbi algorithm a probability of generation with a value smaller than the values that are assigned to the gestures the HMM is supposed to classify, so the definition of limits to the acceptable values of probabilities that a certain HMM is assigned regarding the sequences actually representing the gestures that HMM is supposed to classify, can reduce the number of false positives during the gesture recognition phase, as Prekopcsák ([Pre08]) suggests.

In the end, an Android application that serves as a proof of concept was developed. The goal of this proof of concept application is to use the software framework developed to recognize gestures, and so, it was a very simple application. It served to include the framework files in the project where the application was being developed, and make use of the functionalities made available to display on the smartphone screen the name of the gestures identified.

3.5 Conclusions

The main goal of the current work is to develop a software framework to be used in Android applications, that spares the developer from having to implement the gesture training and recognition processes.

The main challenges identified during the state of the art review are the gesture recognition itself, the user-independent recognition, the usage of different sensors with different sampling rates and with noise, and the fusion of signals from the different sensors, the normalization and the compression of the gesture samples as strategies to increase the recognition accuracy.

To deal with the challenge of achieving a user-independent recognition and to implement the recognition of the gestures, the HMM approach was chosen, once that it is a statistical method, and so, it can enable the achievement of a user-independent gesture recognition, which will render Android applications that incorporate the framework capable of recognizing gestures from many users and will not oblige the user to learn how to exactly perform a gesture. User-independence is achieved when the gesture models are flexible enough to deal with differences on the gesture samples they are supposed to recognize and is this flexibility that enables the non-exact gestures and multiple user gesture recognition.

The software framework will avoid any Android application user to spend any effort training the gesture models, as the framework allows the models to be pre-trained, and then embedded in the Android application.

Once that there is no standard basic gesture definition to rule the hand gestures, as there is an alphabet to rule speech or writing, a set of possible gestures must be chosen in order to test and to be embedded in the software framework. The objective of the Dissertation developed was not to define a gesture vocabulary, so it was used one the already defined and found during the state of the art review ([[KKM⁺05](#), [LWZ⁺09](#)]).

Chapter 4

A Tool for Gesture Recognition

In this Chapter, the details about the proposed framework are presented. The aim of that framework is to facilitate the development of Android applications that explore gestures a means of interaction with the application user. The most important functionalities are identified and we expose how they were implemented.

4.1 Functionalities Identified

In order to implement the functionalities related to HMMs, a Java library available online, named jahmm, was used ([Fra]). This Java library implements, among others, the Segmental K-means and the Baum-Welsh algorithms used to train the Hidden Markov Models (HMMs) and the Viterbi algorithm, used to calculate the probability of generation of a given sequence of observations, in this case a sequence of readings from the smartphone's sensors, given a HMM.

One of the objectives is to embed in the framework some HMMs, in order to spare the framework users the effort of generating HMMs to recognize gestures each time they want to develop a new Android application, and so, the HMMs concerning those gestures must be generated before they can be used by the framework. Although the gesture recognition is meant to be performed in an Android powered smartphone, the HMM are not generated on a smartphone but on a computer. The jahmm library allows the HMMs that were trained using samples collected from many people to be transposed into text files, and also allows the inverse operation of reading files containing descriptions of HMM and create HMM from the files' contents. This functionalities available in the jahmm library were used to generate the HMMs on a computer, and transport them to any Android application that uses the proposed framework to recognize gestures. The gestures trained during the development of the current work are embedded in the proposed framework and ready to use, but they can be changed, if any developer feels the need to include other gestures, with the difference that if this is the case, new HMMs must be trained. In any of the cases, the Android developer who wishes to include the proposed framework in an Android application, does not need

to implement the recognition, nor the HMM generation processes. The HMM generation module, which is meant to be run in a computer, receives several samples representing the gestures to be trained in the form of text files containing the values collected from the smartphone's sensors, and uses the samples to estimate the parameters of each HMM using the K-means and Baum-Welsh algorithms. After that, each HMMs' parameters are written into a file, which is made available to the user of the HMM generation module. The gesture recognition module, meant to be included in Android applications, uses the jahmm library functionalities and takes the files containing HMM descriptions generated by the HMM generation module, and builds HMMs from those files when the Android app is running. For instance, the gestures presented on 3.3 were trained and the files containing the HMMs' parameters generated, and if a developer wishes to embed those gestures in an Android application, all he has to do is include the gesture recognition module in the Android application, together with the files containing the parameters of the HMMs, and specify the actions that must be triggered when the gesture recognition module reports the detection of one of the gestures of the alphabet. But if for some reason, is necessary to add more gestures, the developer is also allowed to do that, since the HMMs meant to recognize those gestures are generated.

Although none technique to obtain orientation using sensor fusion was used, some transformations on data coming from the sensors were employed, and if new models are being generated, this must be taken into account. The data from the sensors can be normalized so that all the values on every sample are between $[-1, 1]$. Also the sample can be compressed in order to reduce the number of sensor readings within it to a given number ([KKM⁺05]). If new models are being generated, the usage, or not, of these transformations must be explicitly defined in the HMM generation module. The effects of these two transformations are discussed in Chapter 5.

The functionalities identified, considering the hypothesis of allowing the generation of new HMMs are the following:

1. gesture sample compression
2. gesture sample's values normalization
3. training the Hidden Markov Models (HMMs)
4. HMMs testing
5. classification of a gesture, according to the HMMs available

4.2 Implementation of Each Functionality

From an Android application developer point of view, the natural workflow will be as exposed in the previous functionalities enumeration if new gestures must be integrated in the framework, but if no new gestures are to be integrated, only the last functionality will be used: classification of a gesture, according to the HMMs available.

As it was already mentioned, the samples can be compressed, normalized, or both right before they are used to train the HMMs. If the compression of the samples is to be used, the developer

must introduce a hint on the final size of the compressed sample. The goal of the compression is to enable the usage of smaller gesture samples and with the same size during the training phase, in order to achieve a higher recognition accuracy. It must be noted that HMMs can achieve good results even if the samples given have different sizes, but if the sizes of each sample are all similar, the results are better, and also, it should be good if the samples acquired during the recognition phase have a size similar to the size used during the training phase. To compress the samples simple operations are made as follows. Because the size of the sample must obviously be an integer number, as the size of the gesture sample represents the number of sensor readings inside it, the integer division of the size of the sample and the desired final size hint given is made, and the result is called the "interval width". This name was chosen after the usage that it is given to the integer number it represents: it defines how many values from the original sample will be put into a single value on the compressed sample. The average of the values on the original sample will be put into the new one. If the minimum number possible on the interval width is one and occurs if the sample has a smaller or equal size compared to the desired size.

The value normalization is also a set of simple operations. The values reported by a sensor are 3 dimensional, as the sensor reports a value on the x axis, a value on the y axis and a value on the z axis. The gestures are 3 dimensional, so it might happen that one of the characteristics of a gesture is having high values on one axis, and near zero values on the others. As is important to maintain this relation between the several dimensions of the gesture, the highest value of the 3 dimensions is found, and then, all the values on the 3 dimensions of the gesture are divided by that highest value. Let's think of a gesture which is a pure translation from left to right, with the screen of the smartphone pointing upwards, and the top of the smartphone pointing to the front. According to the axes description on [\[Andb\]](#) this gesture will have near zero values on the y axis, near 9.8 values on the z axis and other values on the x axes, according to the acceleration imposed to the smartphone during the gesture. For calculation convenience, let's suppose these values range in the interval $[-5, 5]$, the values on y are really 0, and that the value of the gravity is 10. So the highest value on this hypothetical sample is 10. After the normalization, the y axis will still have zero values ($0/10=0$), the z axis will have values equal to 1 ($10/10=1$) and the x axis will have values between $[-0.5, 0.5]$ ($\pm 5/10=\pm 0.5$), thus, the signal characteristic across the different axes was maintained.

After compression and normalization of the samples, the HMMs training takes place. As there may be many gestures to train and the training operations can take some time to complete, the training is performed with a thread pool, having as many threads available as the cores of the used central processing unit, in order to parallelize each gesture model training. Each thread executes as follows. The gesture samples regarding a gesture are loaded and are given into an implementation of the Segmental K-Means algorithm in order to estimate initial parameters to each model. After having a HMM containing initial parameters, this model is refined by an implementation of the Baum-Welch algorithm. It is important to clarify that the jahmm library makes available an implementation of this algorithm ready to receive big samples, that uses a scaling feature to avoid underflows on its internal calculations. After this, the training process of this gesture is finished

and the thread executing is free to train other gesture, if there is still any gesture waiting for a thread to train it.

After the gesture models training, the testing takes part. The tool implemented picks the samples inside the testing division made earlier and classifies them using the generated models. For each gesture sample, the models are asked which is the probability of them generating that sample, using the Viterbi algorithm. It is returned the probability of each model, and in the end of each model test, it is calculated the recognition accuracy. The information about the test is saved into a text file containing information about the training configuration, the identification of models (gestures) that were tested, and for each model, the identification of the samples used during the test, and for each sample used, the probability value given by all the models¹. This values are used in order to obtain the range of probability values at which each gesture is correctly classified. This range will be used later, on the Android applications, to discard false positives. During the classification phase, each model return the probability of generation of a certain sequence, which may be a not null value if the gesture does not belongs to the gesture alphabet. In case the gesture does not belong to the gesture alphabet, and is classified with a non zero value of probability, this value should be smaller than the values that are returned by the most probable model on a correct classification, so it is important to use the range of probability values at which each gesture is correctly classified, to verify if a gesture was correctly identified.

After the models are generated and tested, the Android developer can use them on Android applications, by inserting them on the assets directory. When using the software framework developed, the Android developer must initialize the framework, and one of the parameters needed for this initialization is the path to the directory where the model description files are. The specification of this path is useful, for instance, if the developer needs to insert more files inside the assets directory, and so, the model description files can be arranged inside their own directory.

It is important to clarify that the recognition is made on a thread other than the Android application main thread, in order not to block whatever an Android application must do, besides identifying the user gestures.

Given that the recognition is made on it's own thread, it is mandatory to have a mechanism that allows the recognition thread to make available the recognition results. In order to enable the recognition thread to do this, an architecture following the Observer Pattern was implemented. It was implemented a Java interface on which is declared a method to be called when a gesture is identified, and this method must be implemented inside one of the classes pertaining to the Android application that is integrating this framework. The header of this method is *onGestureDetected(GestureEvent gEvent)* and the name of the Java interface is *GestureDetectionListener*. The implementing class must implement this interface and also, must be registered on the gesture detection thread in order to be notified, via the implemented interface, of any gesture occurrence. The gesture detection thread keeps the registered observers, and when a gesture is detected, the observers are notified. Notifications are made by the gesture

¹An example of one of these files can be found in Appendix A.

detection thread by calling the method *onGestureDetected(GestureEvent gEvent)* of each registered observer. An observer is allowed to register and unregister at any time. Registration is made with *addListener(GestureDetectionListener gdl)* and unregistration is made with *removeListener(GestureDetectionListener gdl)*.

The functionalities made available to an Android application developer allow two modes on the gesture recognition. If the developed Android application user is not supposed to provide any feedback about the gesture's beginning or ending, the thread keeps executing and saving all the values reported by the sensors needed to perform the gesture recognition, and periodically verifies if a gesture occurred. Otherwise, if the developed Android application user is meant to sign the beginning and the ending of a gesture, then the thread is not running, and the developer must use the detection, sensor registering and unregistering, and sensor data reset methods properly.

It is important to expose that the gesture recognition thread saves the values from the sensors in order to use them on each gesture identification attempt, but not all the values reported by the sensors are kept. Based on the average gesture duration (that can be identified computing the average duration of the gestures used to train the models) only the values that were reported within the last window of time with that duration are reported. And as it was already referred, the gesture detection thread periodically verifies if a gesture occurred. The time between each verification is also configurable, and if a gesture was identified, all the values that were already received from the sensors are discarded. If a gesture occurred and it was identified and reported, the stored values are not useful anymore, and the attention is drawn to the values to come, in order to identify the next gestures.

All the necessary methods are available through an Application Programming Interface (API) and are detailed as follows. One of the first methods that should be used is the method *init(Context appContext, String gestureTemplatesPath)* which initializes the framework configurations, such as average gesture duration and the duration of the interval between each recognition attempt, declares the sensors used to generate the Hidden Markov Models within the Android application, making the application Context² and the path to the directory containing the files describing the models of the gestures available to the framework, and loading the gesture modules from the files to the smartphone's memory. As the Observer Pattern suggests, two methods to perform registration and unregistration of listeners within the gesture detection thread were implemented, and their headers are *addListener(GestureDetectionListener gdl)* and *removeListener(GestureDetectionListener gdl)*. A *start()* method was implemented in order to trigger the beginning of execution of the gesture detection thread and to register the sensors needed (the *init(Context appContext, String gestureTemplatesPath)* method only prepares everything the thread needs to be started), and a *stop()* method stops the execution of the gesture detection thread and unregisters the sensors used. If the aim of the Android application that is being developed is to perform a detection without any segmentation introduced by the application user, the Android developer needs only to use the exposed methods and periodically, the gesture detection thread

²Refer to the Android Reference [[Anda](#)] to obtain more info about Context

will verify if a gesture occurred. Otherwise, if the Android application is meant to detect gestures on the user command, the developer can associate the following methods, for instance, to a button, and the user may signal the beginning and the end of a gesture. The method *registerSensorEventListeners()* registers the sensors to be used on gesture recognition, and the opposite is done with *unregisterSensorEventListeners()*. After the sensors are registered, the values they report are saved within the gesture detection thread. When the user finished the gesture and, for instance, releases a button, the developer should use *unregisterSensorEventListeners()* to stop using the sensors, *detectGesture()* to identify which gesture was performed (the result will also be returned through the *onGestureDetected(GestureEvent gEvent)* method implemented), and *resetSensorsData()* in order to discard the values regarding the last gesture.

4.3 Conclusions

The jahmm[Fra] library was used, and thus the implementation of the identified functionalities was faster. Once the Android applications developer has all the samples needed to train the gestures, every step towards getting the files describing the trained HMMs is automated to the maximum, and so is every task, regarding the gesture recognition on the developed Android application, after the development phase. In order to improve recognition accuracy, if necessary, it is possible to compress all the gesture samples collected to generate the HMMs, and also to normalize all the values within each sample. The recognition of gestures might be according to the developed Android application user feedback on the beginning and ending of the gesture, or can be made continuously within a separate thread.

Limits for the range of values of probabilities at which a gesture is correctly classified by a HMM are obtained during the tests phase, in order to allow the identification of false positives during the gesture recognition phase on an Android application.

Chapter 5

Tests And Results

5.1 Setup

In order to test the Hidden Markov Models generation and sample classification models developed, samples of each gesture contained in the gesture alphabet used (see Section 3.3) were collected from several subjects, using a Samsung Galaxy Nexus smartphone, running Android 4.0.1 platform, given that a sample is considered to be the collection of the sensors' readings that were collected while a gesture was performed, and the sensor readings are the values reported by a sensor at a time. These values are the timestamp at which the values were read by the sensor, the value read on the x axis, the value read on the y axis and the value read on the z axis. The samples were collected using the maximum possible sampling rate, using the `SensorManager.SENSOR_DELAY_FASTEST`, of the Android's `SensorManager` class [Andb]. This allowed to study the effect of different sampling rates on gesture recognition later, without collecting more samples. This effect was studied by considering, on each sample, only the readings that could possibly be obtained in a given sampling rate, below the maximum sampling rate achieved on the samples collection. This smartphone uses MEMS inertial sensors from Invensense [inv] and these sensors come inside a single device, a motion tracking device, that combines an accelerometer, a gyroscope and a magnetometer. This motion tracking device from Invensense implements sensor fusion and processing at hardware level, and thus that is not necessary at application level. The motion tracking device present in the Samsung Galaxy Nexus smartphone makes available some virtual sensors, that is, sensors that are not physically available inside the smartphone, but are simulated by applying filters and fusion to the signals coming from the physically available sensors. The sensors whose vendor is Invensense that the Android API identifies are (the names used here are the names obtained by the Android API when querying the operating system about the available sensors):

- Gyroscope - detects angular velocity

Tests And Results

- Accelerometer - detects acceleration due to gravity and to movements of the smartphone relative to the earth.
- Magnetic Field - measures the strength of magnetic fields
- Orientation - measures roll, pitch and azimuth
- Rotation Vector - measures the rotation of the smartphone on each of the 3D axes relative to the earth
- Linear Acceleration - measures only acceleration due to movements of the smartphone - acceleration due to gravity is filtered out
- Gravity - measures only the acceleration due to gravity

For more details about the sensors on Android devices refer to [\[Andb\]](#).

From the exposed list of sensors, the Rotation Vector, the Linear Acceleration and the Gravity sensors are examples of virtual sensors, available through filtering and fusion of the physically available Gyroscope, Accelerometer and Magnetic Field sensors.

Another advantage of the Invensense hardware is that all the sensors used report new values at the same rate. The smartphone used has more sensors besides the ones already referred¹, but because they do not fit in the context of the current work, they were ignored. In order to verify the impact of the usage of more or less sensors at the same time, on the sampling rate, all the possible combinations of the seven previously referred sensors were experimented, and the sensors were used with the maximum sampling rate possible². It was verified that for every possible combination of the referred sensors, the report rate of all the sensors is the same, and close to 100Hz. The sampling rate can vary on ± 2 Hz from sample to sample, but within the same sample, all sensors report with the same sampling rate, in the interval of about 98hz to about 102Hz.

This fact avoided the inclusion of a technique to combine readings from sensors reporting at different rates, and thus, dealing with different amounts of data from each sensor. In the case of not having the same sampling rate, the sample compression that will be referred next could be used, and the length of each sensor's sample that provides the best recognition accuracy could be obtained experimentally.

The collection of samples was made in a period of 5 days, achieving a number of 43 samples per gesture, and during each session, the gestures were stored while holding the smartphone in three different positions, with the goal of study the impact of including the same gesture in various orientations in the gesture alphabet. For the first position, the subjects held the smartphone in a position parallel to the ground, with the screen facing up and the top of the smartphone pointing forward, as illustrated in Figure 5.1(a). For the second position, the subjects held the smartphone in a position perpendicular to the ground, with the top of the smartphone pointing to the left, as

¹Other sensors are Light, Proximity and Pressure sensors

²It is possible, due to the functionalities implemented on Android API, to give a hint on which should be the sampling rate of the sensors. For more detail, refer to SensorManager class on Android API Reference [\[Anda\]](#)

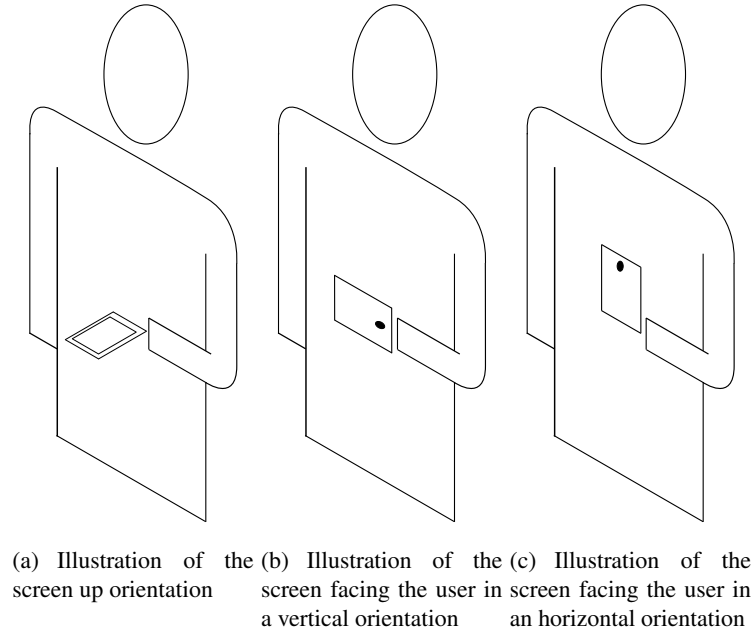


Figure 5.1: Illustration of the three orientations of the smartphone while collecting gesture samples

illustrated in Figure 5.1(b). For the third position, the subjects held the smartphone in a vertical position, perpendicular to the ground, with the smartphone's screen pointing to themselves and the top of the smartphone pointing up, as illustrated in Figure 5.1(c). To use the gestures in more than one orientation, one model per gesture per orientation was trained. As there are 8 gestures in the alphabet and each gesture was made in three orientations, there is a total of 24 gestures, meaning that 24 HMMs were trained. The gesture alphabet was composed of 8 gestures identified by a Nokia research (see Chapter 3.3) and can be found on [KKM⁺05].

The collected samples were then mixed up, in order not to distinguish samples from different users, once the objective is user independent recognition, and this set of mixed samples was divided into two sets five times. On each division of the total set of samples, the first set, with 75% of the samples, was used to train the models, while the second set, with 25% of the samples was used to test the models. This division of the total set of samples was made 5 times in order to allow the comparison between various combinations of samples. These sets of samples were used to train and test the system developed using a computer, once the processing capacity of a computer is greater, and thus the tests finish faster.

In order to comprehend which is the best way to use the data collected from the smartphone's sensors in the gesture recognition and HMM generation modules developed, some experiments were performed, allowing the transformation of the data and the understanding of how the system reacts to different representations of the same gesture. One of the transformations made was the normalization of the sensor readings values on each sample, and the other transformation was the compression of the samples. The samples collected from test subjects were collected at about 100Hz, given that this value was not affected by the sensors used, due to processing made to

the sensors' readings at hardware level. The usage of these transformations was used as follows. First, the samples were used without any of these transformations, that is, the data was used as it came from the sensors. This is called raw data. Then, only the normalization of the values of the samples between $[-1, 1]$ was used on raw data. After that, only compression was used on raw data. Finally, normalization and compression were both applied to raw data. It is important to clarify that according to the samples collected to train and test the models, the average duration of the gestures was 1.7 seconds. As the sampling rate of the sensors when using the developed software framework drops to 30Hz, an average gesture will be composed of about 50 sensor readings. This number of sensor readings was used during the tests as a hint on the final size of the samples when compression was used.

It was also verified that when using the developed framework, which uses the sensors on a separate thread, the sensors' sampling rate drops to about 30Hz. It is believed that this drop is caused by the fact that the values from the sensors are being received by a thread other than the main application thread where, for instance, the user interface is designed on the smartphone's screen.

Another test was performed not at the data level, but at the HMMs level. This test consisted in using the same datasets, and data transformations, but varying the number of states of the HMMs. The goal of this test was to measure the impact on the recognition accuracy, by using a different number of states.

The inertial sensors available on the smartphone used, that are useful to gesture recognition are the gyroscope, the accelerometer and the linear acceleration sensor. As the main goal of the realized work was to use the accelerometer and the gyroscope to improve gesture recognition, by using information regarding the translation and rotation of the smartphone during the execution of the gesture, the first set of tests was made using these two sensors. The second set of tests was performed using the gyroscope to obtain information on rotation, but instead of using the accelerometer to obtain information on translation, it was used the linear acceleration sensor. It is expected to have some misclassifications due to the absence of gravity (this may lead to a gesture made in an orientation being classified as the same gesture but on a different orientation). The third set of tests was made using only the accelerometer, to enable the comparison of results between the recognition accuracy using one sensor, and the recognition accuracy using two sensors.

Also, and as it is proposed to study the impact of having the same gesture performed with the smartphone on different orientations, the tests were performed using gestures from all orientations all together, and then using gestures on each orientation separately.

5.2 Results

In this section we present the most significant results that were obtained regarding smartphone gesture learning, from a user-independent point of view.

5.2.1 Using the Accelerometer and the Gyroscope

Considering the usage of both the accelerometer and the gyroscope, the mean recognition accuracy considering all the five datasets, using the gestures on the three mentioned directions was as exposed in table 5.1.

Table 5.1: Average recognition of the five datasets using all directions, using the accelerometer and the gyroscope.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	40.9%	44.5%	43.9%	44.3%
Normalization only	82.1%	80.5%	80.3%	79.4%
Compression only	81.0%	83.5%	83.3%	84.9%
Normalization and compression	84.3%	83.9%	85.1%	84.6%

As it can be observed, the greatest value for the overall average recognition accuracy is 85.1% when both sample normalization and compression are used and the number of HMM states is 8. Using only normalization, a recognition accuracy of 82.1% is achieved when using 4 HMM states and when using only compression, an average accuracy of 84.9% is achieved when using 10 states.

The recognition accuracy when the smartphone was parallel to the ground with the screen facing up and the top pointing forward, using the accelerometer and the gyroscope, was as exposed in table 5.2.

Table 5.2: Average recognition of the five datasets using the smartphone with it's screen up, using the accelerometer and the gyroscope.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	58.1%	61.3%	60.6%	61.6%
Normalization only	88.0%	86.8%	86.0%	86.0%
Compression only	89.5%	90.3%	90.9%	90.8%
Normalization and compression	90.5%	92.0%	91.3%	91.8%

As it can be observed, the highest average recognition accuracy was achieved using sample normalization and compression and 6 HMM states, with a value of 92.0%. Using only sample normalization the higher recognition accuracy was achieved using 4 HMM states and has a value of 88.0%. Using compression only a value of 90.9% was achieved when using 8 HMM states. Using no sample compression nor normalization, the maximum recognition accuracy was 61.6% using 10 HMM states.

The recognition accuracy when the smartphone was perpendicular to the ground with the screen facing the user and the top pointing up, using the accelerometer and the gyroscope, was as exposed in table 5.3.

Tests And Results

Table 5.3: Average recognition of the five datasets using the smartphone's with it's screen vertical, facing the user, using the accelerometer and the gyroscope.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	38.0%	42.0%	40.0%	41.8%
Normalization only	81.0%	77.0%	79.0%	76.5%
Compression only	78.3%	82.0%	82.0%	84.3%
Normalization and compression	82.8%	81.3%	84.0%	83.8%

It can be observed that the highest average recognition accuracy was achieved using sample compression only and 10 HMM states, with a value of 84.3%. Using sample compression and normalization, a value of 84.0% was achieved using 8 HMM states. Using only sample normalization an average accuracy of 81.0% was achieved using 4 HMM states. Using raw samples, the highest value of the average recognition was achieved using 6 HMM states, and has a value of 42.0%.

The recognition accuracy when the smartphone was perpendicular to the ground with the screen facing the user and the top pointing left, using the accelerometer and the gyroscope, was as exposed in table 5.4.

Table 5.4: Average recognition of the five datasets using the smartphone with it's screen horizontal, facing the user, using the accelerometer and the gyroscope.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	39.3%	41.8%	41.8%	40.5%
Normalization only	77.3%	77.8%	75.8%	75.8%
Compression only	75.3%	78.3%	77.3%	80.0%
Normalization and compression	79.8%	78.5%	80.0%	79.7%

Table 5.4 shows that the highest value of average recognition accuracy is 80.0% using either 8 HMM states, normalization and compression, and using 10 HMM states and sample compression. Using only sample normalization, an average accuracy of 77.8% was achieved using 6 HMM states and raw samples provided an average accuracy of 41.8% using 6 and 8 HMM states.

5.2.2 Using the Linear Acceleration Sensor and the Gyroscope

Considering the usage of both the linear acceleration sensor and the gyroscope, the mean recognition accuracy considering all the five datasets, using the gestures on the three mentioned directions was as exposed in table 5.5.

Table 5.5 exposes that the highest value of the average recognition accuracy was achieved when using sample compression and normalization and 6 HMM states. Using sample compression only and 8 HMM states, an average accuracy of 89.0% was achieved and using sample normalization only, an average recognition accuracy with a value of 89.8% was achieved using 8 HMM states. Using raw data, an average accuracy of 52.9% was achieved with 10 HMM states.

Tests And Results

Table 5.5: Average recognition of the five datasets using all directions, using the linear acceleration sensor and the gyroscope.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	44.8%	49.5%	52.7%	52.9%
Normalization only	89.2%	88.9%	89.8%	88.6%
Compression only	84.2%	87.1%	89.0%	88.0%
Normalization and compression	89.1%	90.6%	90.0%	89.7%

The recognition accuracy when the smartphone was parallel to the ground with the screen facing up and the top pointing forward was as exposed in table 5.6.

Table 5.6: Average recognition of the five datasets using the smartphone with its screen up, using the linear acceleration sensor and the gyroscope.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	54.6%	58.3%	63.3%	62.3%
Normalization only	96.3%	96.0%	95.8%	94.7%
Compression only	95.5%	94.1%	95.8%	93.1%
Normalization and compression	97.8%	97.5%	97.0%	95.8%

It can be observed that the highest recognition accuracy achieved is 97.8% using 4 HMM states, sample compression and normalization. Using only sample compression, an average recognition accuracy of 95.8% was achieved using 8 HMM states and using only sample normalization, an average accuracy of 96.3% was achieved using 4 HMM states. With raw data, using 8 HMM states an average accuracy of 63.3% was achieved.

The recognition accuracy when the smartphone was perpendicular to the ground with the screen facing the user and the top pointing up was as exposed in table 5.7.

Table 5.7: Average recognition of the five datasets using the smartphone's with it's screen vertical facing the user, using the linear acceleration sensor and the gyroscope.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	45.8%	52.0%	55.5%	56.3%
Normalization only	92.3%	91.8%	93.0%	92.5%
Compression only	86.8%	88.8%	92.3%	88.8%
Normalization and compression	93.3%	93.8%	93.8%	93.8%

It can be observed that, using sample normalization and compression with either 6, 8 or 10 HMM states. Using only sample compression, an average recognition accuracy of 92.3% was achieved with 8 HMM states, and using normalization only, an average recognition accuracy of 93.0% was achieved using 8 HMM states. Using raw data, an average recognition accuracy of 56.3% was achieved using 10 HMM states.

Tests And Results

The recognition accuracy when the smartphone was perpendicular to the ground with the screen facing the user and the top pointing left was as exposed in table 5.8.

Table 5.8: Average recognition of the five datasets using the direction with the smartphone's screen horizontal facing the user, using the linear acceleration sensor and the gyroscope.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	45.5%	48.0%	50.8%	51.8%
Normalization only	93.3%	90.3%	90.8%	90.8%
Compression only	84.0%	87.5%	89.0%	88.5%
Normalization and compression	93.5%	93.3%	93.5%	90.0%

The table 5.8 exposes that the higher value of average recognition accuracy was achieved using sample normalization, compression and either 4 and 8 HMM states, and is 93.5%. Using sample compression only, an average recognition accuracy of 89.0% was achieved using 8 HMM states and using normalization only an average recognition accuracy of 93.3% was achieved using 4 HMM states. Using raw data and 10 HMM states an average recognition accuracy of 51.8% was achieved.

5.2.3 Using only the Accelerometer

Considering the usage of only the Accelerometer, the mean recognition accuracy considering all the five datasets, using the gestures on the three mentioned directions was as exposed in table 5.9.

Table 5.9: Average recognition of the five datasets using all directions, using only the accelerometer.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	30.5%	34.3%	35.3%	36.7%
Normalization only	44.2%	46.1%	48.3%	51.6%
Compression only	56.1%	61.5%	62.7%	64.4%
Normalization and compression	50.8%	54.7%	58.1%	60.7%

It can be observed that the higher average recognition accuracy has the value of 64.4% and was achieved using sample compression only and 10 HMM states. Using sample compression and normalization the higher average recognition accuracy was achieved using 10 HMM states and has a value of 60.7%. Using sample normalization, the average recognition accuracy reached the value 51.6% using 10 HMM states, and raw data provided an average recognition accuracy of 36.7% using 10 HMM states.

The recognition accuracy when the smartphone was parallel to the ground with the screen facing up and the top pointing forward was as exposed in table 5.10.

It can be observed that the highest recognition accuracy is achieved using 10 HMM states and sample compression, and has a value of 73.8%. Using sample normalization and compression, the

Tests And Results

Table 5.10: Average recognition of the five datasets using the smartphone's with it's screen up, using only the accelerometer.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	32.5%	41.3%	44.3%	45.5%
Normalization only	50.8%	47.8%	53.2%	52.5%
Compression only	65.0%	71.0%	72.0%	73.8%
Normalization and compression	57.5%	59.0%	65.5%	66.5%

higher recognition accuracy was achieved using 10 HMM states and has a value of 66.5%. Using sample normalization only, the highest average recognition accuracy has the value of 53.2% with 8 HMM states. Using raw data and 10 HMM states, an average recognition accuracy of 45.5% was achieved.

The recognition accuracy when the smartphone was perpendicular to the ground with the screen facing the user and the top pointing up was as exposed in table 5.11.

Table 5.11: Average recognition of the five datasets using the smartphone's with it's screen vertical, facing the user, using only the accelerometer.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	34.8%	35.6%	34.8%	34.3%
Normalization only	45.3%	49.8%	49.8%	54.5%
Compression only	53.8%	59.3%	61.8%	64.0%
Normalization and compression	48.3%	56.3%	56.8%	64.0%

Table 5.11 exposes that the highest value for the average recognition accuracy is 64.0% using 10 HMM states and sample compression only or sample compression and normalization. Using sample normalization only, the highest value of recognition accuracy was 54.5% with 10 HMM states and using raw data, an average recognition accuracy of 35.6% was achieved with 6 HMM states.

The recognition accuracy when the smartphone was perpendicular to the ground with the screen facing the user and the top pointing left was as exposed in table 5.12.

Table 5.12: Average recognition of the five datasets using the smartphone's with it's screen horizontal, facing the user, using only the accelerometer.

	4 HMM states	6 HMM states	8 HMM states	10 HMM states
Raw data	26.5%	27.5%	27.8%	31.3%
Normalization only	36.8%	40.8%	42.0%	48.0%
Compression only	49.5%	54.3%	54.3%	55.8%
Normalization and compression	46.8%	48.8%	52.0%	51.8%

It can be observed that an average recognition accuracy of 55.8% was achieved using sample

compression and 10 HMM states. Using sample compression and normalization, the highest average recognition accuracy value achieved was 52.0% using 8 HMM states, and using raw data, an average recognition accuracy of 31.3% was achieved.

5.3 Result Analysis

As it can be observed in the tables 5.1 to 5.12, using the raw data from the sensors always causes the recognition accuracy to be much lower than using sample compression, normalization, or both. Sample normalization causes the data to be more homogeneous, and does not demand the models to be so generic, in order to deal with samples whose values are within different ranges. The same applies to sample compression.

For all cases, except when the sensors used were the accelerometer and the gyroscope and the smartphone was in a vertical position, facing the user (table 5.3), and when using the accelerometer (tables 5.9, to 5.12), the highest average recognition accuracy is achieved when using both sample normalization and compression.

When using the accelerometer only, the highest average recognition accuracy is achieved using sample compression and 10 HMM states. It can be observed that the average recognition accuracy, except for one case (table 5.11 with raw data, table 5.10 with normalization only), increases by some percentage points when the number of HMM states increases as well, suggesting that the usage of one sensor's signal needs a bigger number of HMM states to be more accurate.

It can be verified that for each sensor combination, the gesture alphabet that achieved the highest average recognition accuracy was the alphabet where only the gestures performed with the smartphone parallel to the ground with it's screen facing up, were included. When the gesture alphabet was composed only of gestures performed with the smartphone's screen facing the user, in an horizontal orientation, the highest average recognition accuracy, within each sensor combination was the lowest and the alphabets including only the gestures performed with the smartphone's screen facing the user in a vertical orientation produced average recognition accuracies between the ones produced by the other two alphabets. This suggests that somehow, the test subjects performed the gestures in more similar manners when using the smartphone with it's screen pointing up, parallel to the ground.

When considering the usage of the accelerometer and the gyroscope simultaneously, and the usage of the accelerometer only, the highest average recognition accuracies of the alphabet composed of gestures from all the orientations are somewhere between the highest average recognition accuracy, and the lowest average recognition accuracy achieved by the alphabets with gestures from only one direction. That is, the highest average recognition accuracy was achieved using the alphabet with gestures made with the smartphone's screen facing up, and the lowest average recognition accuracy was achieved when using the alphabet composed of gestures performed with the smartphone's screen facing the user in an horizontal orientation. The average gesture recognition of the alphabet composed of gestures on all orientations was between these two values, and nearer

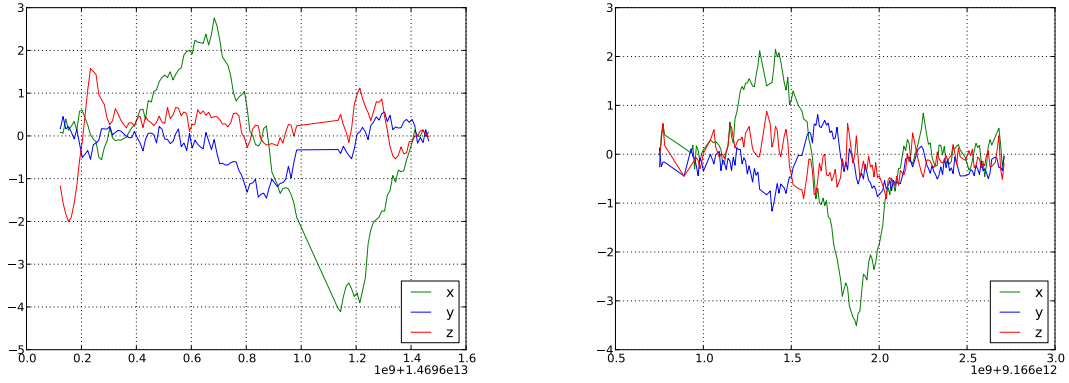
of the values from the alphabet composed of gestures performed with the smartphone's screen facing the user in a vertical orientation, which is also between the values from the up and horizontal alphabet. As the alphabet containing all gestures is composed by mixing the samples from the other three alphabets, this suggests that the gestures on the alphabet with all the orientations are not misclassified because there are gestures from several orientations, but because the models that are supposed to classify the gestures made with the smartphone's screen facing the user, either on a vertical or horizontal orientation, do not produce so good results as the models that are supposed to classify gestures made with the smartphone's screen parallel to the ground.

If the combination of linear acceleration sensor and gyroscope is considered, it can be observed that the exposed on the previous paragraph do not apply, and the average gesture recognition on the alphabet containing gestures from all orientations is the lowest of the four alphabets used. Besides the quality of the models that are supposed to classify gestures made with the smartphone's screen facing the user, either it is on a vertical or horizontal position, is not as good as the quality of the models that are supposed to classify gestures made with the smartphone's screen facing up, the models have no way to distinguish between some gestures, due to the absence of information regarding gravity. An example is the gesture that consists of a translation from the left to the right. Considering linear acceleration, this gesture only has significant values on the x axis, either made with the smartphone's screen facing up and pointing forward, as can be seen in Figure 5.2(a), or made with the smartphone's screen facing the user on a vertical position, as can be seen in Figure 5.3(b). Considering the accelerometer, the information of the direction of gravity, on the z or y axis is crucial for the model to be able to distinguish between the two gestures. This suggests that if this type of gestures is meant to be used with the linear acceleration sensor, no more than one model regarding equivalent gestures should be used. Taking as an example the gesture from left to right, there were three HMM trained to identify the gestures left to right in the three different orientations. Regarding the usage of the linear acceleration sensor, only one model should be used to recognize the gesture left to right either with the smartphone's screen facing up, either with the smartphone's screen facing the user on an horizontal orientation.

Figures 5.2(b) and 5.2(a) illustrate the similarity between the same gesture on different smartphone orientations, if the gravity information is not present, and can be compared with figure 5.3(a), where the information about gravity is present. On figures 5.2(b) and 5.2(a) it can be observed that the characteristic of the signal remains the same (local maximum and minimum of the x component of the signal, painted green). From figures 5.3(b) and 5.3(a) it can be observed that there are sets of readings with values near 10 painted with different colors. In figure 5.3(a) the gravity is represented by the readings on z axis, while in figure 5.3(b) there is a set of readings between 8 and 10 painted blue, that is, on the y axis. The readings on z axis mean that the person who performed this gesture did not held the smartphone in a completely perpendicular to the ground position, so the readings on y axis are near 8, and values on z axis are a little higher. On both figure 5.2(b) and 5.2(a), it can be observed that the characteristic of the readings on x axis maintain (see the local maximum and minimum of this axis) due to the gesture's nature.

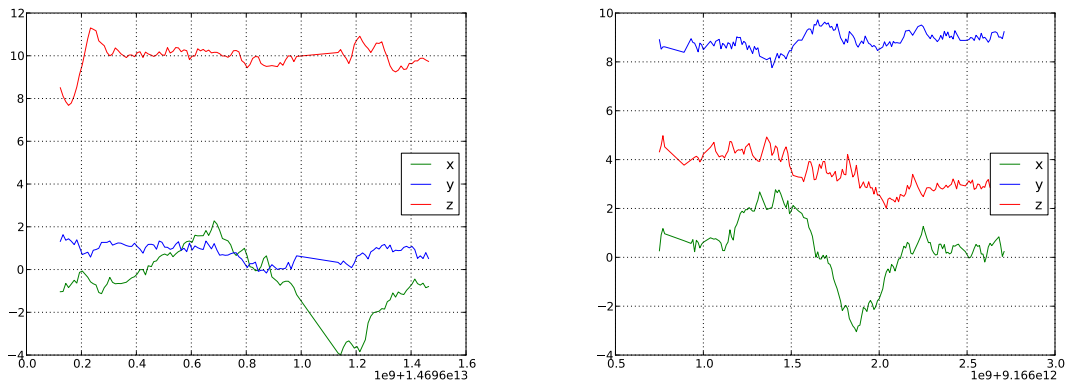
It can be observed that the usage of the accelerometer only, produces average recognition

Tests And Results



(a) Illustration of the signal obtained from the linear acceleration sensor when performing a translation from left to right with the smartphone's screen facing up
(b) Illustration of the signal obtained from the linear acceleration sensor when performing a translation from left to right with the smartphone's screen facing the user

Figure 5.2: Illustration of the similarity between different gestures in the absence of gravity



(a) Illustration of the signal obtained from the accelerometer when performing a translation from left to right with the smartphone's screen facing up
(b) Illustration of the signal obtained from the accelerometer when performing a translation from left to right with the smartphone's screen facing the user

Figure 5.3: Illustration of the differences between different gestures in the presence of gravity

Tests And Results

accuracy values smaller than the usage of the gyroscope with the accelerometer, or the linear acceleration sensor. When considering the alphabet with gestures on all the orientations, the usage of the gyroscope and the accelerometer is better than the usage of the gyroscope and the linear acceleration sensor, due to the presence of similar gestures if the gravity is discarded. Using the gyroscope and the accelerometer, an average recognition accuracy of 85.1% was achieved using 8 HMM, sample normalization and compression, and using the gyroscope and the linear acceleration sensor, an average recognition accuracy of 90.6% was achieved on alphabets with gestures on all orientations. Considering an alphabet composed of gestures on a single orientation, the alphabet composed of gestures performed with the smartphone's screen facing up produces the highest average recognition accuracy using sample compression and normalization, and using 4 HMM states, with a value of 97.8%.

It can also be concluded that the usage of two sensors may be a technique to reduce the HMM training effort, since that the usage of the accelerometer alone produces recognition accuracy values lower than the recognition accuracy achieved with the usage of gyroscope and accelerometer, or gyroscope and linear acceleration sensor.

Considering now all the sensor combinations used, and the gesture alphabets composed only of gestures performed on one orientation, it can be observed, as already mentioned, that the alphabets composed of gestures performed with the smartphone's screen facing the user produce lower average recognition accuracies, than the recognition accuracies produced by the alphabets composed of gestures performed with the smartphone screen facing up, even if all the gesture models on any alphabet were generated using the same number of samples. Besides it is considered that the gestures performed with the smartphone's screen facing the user, either on a vertical or horizontal orientation, were subject of more variation, it is also considered that the number of samples used to generate the models was low. There were collected 43 samples per gesture, and from these 43 samples, 33 were used to train the models and 10 were used to test. It is supposed that, as can be seen on the state of the art review, as the HMM strategy is based on statistical methods, if there were more samples available, the generated models would be more accurate.

Other fact that must be exposed, is that during some of the training attempts performed with the various configurations already mentioned, the jahmm library was not able to generate models due to lack of training samples. As the samples were not enough, it was not possible to extract the characteristics of the models from the samples available. This failure occurred while executing tests with the following configurations:

- using the accelerometer and the gyroscope, 10 HMM states:
 - the gestures made with the smartphone screen facing up, and using sample compression on dataset division number 0;
 - the gestures made with the smartphone screen facing the user on a vertical orientation using compression and normalization on dataset division number 1
 - the gestures made with the smartphone screen facing the user on a horizontal orientation using compression and normalization on dataset division number 2

Tests And Results

- using the linear acceleration sensor and the gyroscope, 4 HMM states:
 - the gestures made with the smartphone screen facing up, and using sample compression and normalization on dataset division number 0;
- using the linear acceleration sensor and the gyroscope, 6 HMM states:
 - the gestures made with the smartphone screen facing up, and using sample compression on dataset division number 4;
 - the gestures made with the smartphone screen facing up, and using sample compression and normalization on dataset division number 0;
 - the gestures made with the smartphone screen facing up, and using sample compression and normalization on dataset division number 4;
- using the linear acceleration sensor and the gyroscope, 10 HMM states:
 - the gestures made with the smartphone screen facing the user on a horizontal orientation using compression and normalization on dataset division number 0
 - the gestures made with the smartphone screen facing up using compression on dataset division number 4
 - the gestures made with the smartphone screen facing up using compression and normalization on dataset division number 0
 - the gestures made with the smartphone screen facing up using compression and normalization on dataset division number 4

Once that an alphabet composed of gestures on all orientations is built using the gestures from the three alphabets containing only one direction, the training attempts on the multi-directional alphabets which were built with the mentioned alphabets also failed.

Chapter 6

Conclusions and Future Work

The main goal of this Dissertation was to develop a framework that implements gesture recognition and is meant to be used in Android applications, that should use gestures as a means of interaction with the user. The idea is that when an application that integrates this framework is installed on a smartphone, the user can interact with that application not using the fingers to point or drag objects of the application's graphical user interface, but using gestures, given that a gesture is a movement that the user performs with the hand that is holding the smartphone. Nowadays, the smartphones' capabilities are increasing, either in terms of hardware and software, and their importance in the users' daily life is also increasing, due to the ease of many tasks that the smartphones introduce, and this makes the users have their smartphone close to them very often. Given that one of the capabilities that is increasing is the capability of perceive the smartphone physical state, regarding movements, due to the inclusion of inertial sensors, the opportunity of explore gestures as a means of interacting with the smartphone arises. The usage of gestures as a means of interacting with the smartphone has the advantage of being natural and intuitive, and do not require the user to look at the smartphone in order to check if the right button is being pressed.

Once that a framework like the proposed is meant to be a generic framework, in the sense that it could be integrated in any kind of Android application, and an application usually is meant to be used by several users, the module of the framework that carries out the gesture recognition must be capable to handle the different ways in which different users perform the same gesture. During the state of the art review, statistical approaches appeared to be a promising approach when dealing with recognizing gestures from many users, since these methods can estimate models of the gestures given a set of samples of those same gestures. The property of recognizing a gesture, no matter who is performing it, and recognizing the gesture, even if the person who is performing a gesture to be recognized did not perform any training samples is called user independent recognition. During the state of the art review, the algorithms that showed to be more promising considering user independent recognition were Hidden Markov Models (HMM) and Support Vector Machines (SVM).

Conclusions and Future Work

The HMM was the approach used to carry out gesture learning and recognition and it proved to be a good technique, either achieving the demanded user independent recognition and a high average recognition accuracy.

6.1 Accomplishments

Using the HMM approach it was achieved an average recognition accuracy of 97.8% using an alphabet of 8 gestures and collecting gesture samples with the gyroscope and the accelerometer, and an average recognition accuracy of 85.1% using the gyroscope and the accelerometer on an alphabet of 24 gestures. Sample compression and normalization was used in order to evaluate its impact on average recognition accuracy, and in most cases, the usage of both normalization and compression gave the best results. The usage of raw data did not allow so higher average recognition accuracies, as compared to the usage of sample normalization, sample compression, or both. The HMMs were trained using samples from several people in order to better achieve user independent recognition, and the training phase took place on a computer, where files containing the parameters of each model were generated, and then imported by the gesture recognition module running in an Android application. The gesture recognition module is ready to be used in an Android application, and to recognize the gestures from the alphabet used. This module runs in its own thread in order not to block the main thread of an Android application that incorporates that module, where the user interface is designed.

The impact of the usage of information regarding rotation with translation was studied, and it was concluded that using these two types of information can produce better results than using only information about translation.

As far as could be found in the state of the art review, there is not a framework capable of facilitating the development of Android applications that make use of gestures detected with the smartphone embedded sensors, so this is the main contribution made.

As the proposed framework is capable of dealing with several trained gestures, an application that incorporates the proposed framework could be used to trigger any actions. These actions could be within the scope of the application or could be outside the scope of the smartphone, making of the smartphone a device to interact with the environment. Examples of possible usages within the scope of the smartphone are rotating the smartphone around the x axis to scroll up or down on a list of the files stored on the smartphone, start other application or automatically call a certain person in the contact list. Examples of possible usages outside the scope of the smartphone are controlling Bluetooth enabled electrical devices, and the smartphone could be used as single remote controller to interact with a DVD player, a radio and a TV, without having to press buttons as in a usual remote controller.

6.2 Future Work

In order to continue and improve the work already developed, the following topics are elicited:

Conclusions and Future Work

- as the validation was centered in the Hidden Markov Models generation and utilization during the recognition, the proposed framework should be validated using an Android application with a real purpose, other than a simple proof of concept application
- study the impact of using data quantization [LWZ⁺09], once that this will allow the usage of unidimensional Hidden Markov Models(HMMs) instead of the six-dimensional used
- addition of noise to the training samples, as suggested by Kay [Kay00]
- development of a better technique to perform automatic gesture segmentation on real-time, without causing delays, or sensors' sampling rate drops
- study the impact of including more gesture samples in training than the used on the present work
- study the impact of including information about gravity in the Hidden Markov Models, when using the linear acceleration sensor, that is, besides giving to the model the tree values reported by the gyroscope, and the three reported by the linear acceleration sensor, give three more regarding the gravity component on each of the 3 dimensional axes.
- evaluate the performance of the Support Vector Machines (SVM) algorithm instead of the HMM, once that the state of the art review suggests that this is also a good algorithm, and due to time restrictions, was not used
- study a way of enabling the adaptation of the HMMs in each instance of an application integrating this framework to the application's user

Conclusions and Future Work

References

- [Anda] Android reference. <http://developer.android.com/reference/packages.html>, Accessed: 2013-06-03.
- [Andb] Android sensors overview. http://developer.android.com/guide/topics/sensors/sensors_overview.html, Accessed: 2013-06-03.
- [AV10] Ahmad Akl and Shahrokh Valaei. Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 2270–2273, march 2010.
- [CHL13] Salman Cheema, Michael Hoffman, and Joseph J. LaViola. 3D Gesture classification with linear acceleration and angular velocity sensing devices for video games. *Entertainment Computing*, 4(1):11–24, February 2013.
- [DD96] Rakesh Dugad and U. B. Desai. A tutorial on hidden markov models. Technical report, Signal Processing and Artificial Neural Networks Laboratory Department of Electrical Engineering Indian Institute of Technology — Bombay Powai, Bombay 400 076, India, 1996.
- [Fra] Jean-Marc François. jahmm: an implementation of hidden markov models in java. <https://code.google.com/p/jahmm/>, Accessed: 2013-06-03.
- [HCL03] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.
- [HVL10] Michael Hoffman, Paul Varcholik, and Joseph J. LaViola. Breaking the status quo: Improving 3D gesture recognition with spatially convenient input devices. *2010 IEEE Virtual Reality Conference (VR)*, pages 59–66, March 2010.
- [inv] Mems motionprocessing for smartphones. <http://www.invensense.com/cn/mems/smartphones.html>, Accessed: 2013-06-03.
- [JT11] Derick a. Johnson and Mohan M. Trivedi. Driving style recognition using a smartphone as a sensor platform. *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1609–1615, October 2011.
- [KARY12] Mridul Khan, Sheikh Iqbal Ahamed, Miftahur Rahman, and Ji-Jiang Yang. Gesthaar: An accelerometer-based gesture recognition method and its application in nui driven pervasive healthcare. In *Emerging Signal Processing Applications (ESPA), 2012 IEEE International Conference on*, pages 163–166, jan. 2012.

REFERENCES

- [Kay00] Steven Kay. Can detectability be improved by adding noise? *Signal Processing Letters, IEEE*, 7(1):8–10, jan. 2000.
- [KKM⁺05] Juha Kela, Panu Korpipää, Jani Mäntyjärvi, Sanna Kallio, Giuseppe Savino, Luca Jozzo, and Sergio Di Marca. Accelerometer-based gesture control for a design environment. *Personal and Ubiquitous Computing*, 10(5):285–299, August 2005.
- [LWZ⁺09] Jiayang Liu, Zhen Wang, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uWave: Accelerometer-based personalized gesture recognition and its applications. *2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–9, March 2009.
- [LXC⁺11] Lu Lou, Xin Xu, Juan Cao, Zhili Chen, and Yi Xu. Sensor fusion-based attitude estimation using low-cost MEMS-IMU for mobile robot navigation. *2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference*, pages 465–468, August 2011.
- [PKS⁺13] E. Pulido Herrera, H. Kaufmann, J. Secue, R. Quirós, and G. Fabregat. Improving data fusion in personal positioning systems for outdoor environments. *Information Fusion*, 14(1):45–56, January 2013.
- [Pre08] Zlotán Prekopcsák. Accelerometer based real-time gesture recognition. In *Proceedings of the 12th International Student Conference on Electrical Engineering*, pages 1–5, 2008.
- [Pyl05] Timo Pylvänäinen. Accelerometer Based Gesture Recognition Using Continuous HMMs. *Pattern Recognition and Image Analysis. Second Iberian Conference, IbPRIA 2005. Proceedings, Part I*, 3522:639–646, 2005.
- [Rab90] Lawrence R. Rabiner. Readings in speech recognition. pages 267–296, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
- [Rub91] Dean Rubine. Specifying gestures by example. *ACM SIGGRAPH Computer Graphics*, 25(4):329–337, July 1991.
- [Sal12] Laura Salhuana. Tilt sensing using linear accelerometers. *Freescale Semiconductor, Inc., Application Note, Document Number: AN3461, Rev. 4, 02/2012*, 2012.
- [Vis11] Ingmar Visser. Seven things to remember about hidden Markov models: A tutorial on Markovian models for time series. *Journal of Mathematical Psychology*, 55(6):403–415, December 2011.
- [WW04] Daniel Wilson and Andy Wilson. Gesture recognition using the xwand. Technical Report CMU-RI-TR-04-57, Robotics Institute, Pittsburgh, PA, April 2004.

Appendix A

Sample Containing Test Results

In this appendix it is shown the content of a file obtained from one of the tests performed.

The meaning of the contents on the file name are the following. The file name starts with "s=4" meaning that Hidden Markov Models with 4 states were used on this test. The following element is due to the fact that gestures performed using the smartphone on several orientations. In this example, "testUP" means that only the gestures that were performed with the smartphone's screen facing upwards were used. The next group of 2 numbers refers to the usage of compression and normalization. "00" means that no compression and no normalization were used. The next number refers to the number of the original gesture sample dataset division. This test used gesture samples contained in the division number 0.

The contents of the file are the following. The first line indicates the number of states on the HMMs used during the test. Next to it appears a path pointing to samples of a gesture. The gesture name is the name of the last directory on the path. On this example, the first model to be tested was the model describing a gesture named "Elbow Up". Next to the model identification appears a name of a file used during the test. This is useful to identify the gesture samples used on each test. Although only the name of the files regarding the accelerometer appear, this does not mean that only the files from the accelerometer were used. On this line, only the number is relevant, and only the name of one file is used to reveal which sample was used. To note that other files regarding the same sample begin with the same number. On the following 8 lines appear the probability returned by each model. On the first set of 8 lines on this example, it can be observed that the model with the highest probability of having generated that sample is a model named "Right To Left Up" with a value of 2.9675013270215624E-261. As explained earlier, the first model tested on this example has the name "Elbow Up" so we are in the presence of a misclassification. When all the samples to test each model end, it can be seen how many samples received the proper classification and how many did not. On this example, 4 out of 10 samples were correctly classified. Next to that, the accuracy rate of the test is written, and on the next two lines it can be found the maximum and minimum value of the correct classifications on the test of the current model.

A.1 Results Output

File name: s=4_testUp_00_0.txt

Contents of the file:

states: 4

/home/paulo.silva/SAMPLES/test/qnt/perOrientation/screenUp/Test/0/Elbow Up

5-acc.txt

Right To Left Up 2.9675013270215624E-261

Circle Anticlockwise Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

34-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

6-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

16-acc.txt

Square Up 8.406600127865964E-198

Elbow Up 6.004230346548663E-199

Circle Anticlockwise Up 2.9262530801326123E-214

Circle Clockwise Up 6.452581817685952E-228

Right To Left Up 4.67628004072163E-249

Left To Right Up 4.110567499426504E-252

Sample Containing Test Results

Up Up 7.232003317545294E-277

Down Up 2.1773393633950306E-300

32-acc.txt

Elbow Up 2.6777525623956033E-247

Circle Anticlockwise Up 1.1096294078855808E-251

Square Up 1.9041411534137692E-258

Left To Right Up 5.030938490633799E-294

Circle Clockwise Up 2.08449813035998E-295

Right To Left Up 0.0

Up Up 0.0

Down Up 0.0

9-acc.txt

Elbow Up 5.343865723631202E-228

Square Up 1.8148404087055825E-249

Left To Right Up 2.251292977235336E-260

Circle Anticlockwise Up 1.5602305634171118E-265

Right To Left Up 1.2020304562938819E-287

Circle Clockwise Up 1.085086956E-315

Up Up 0.0

Down Up 0.0

3-acc.txt

Square Up 5.729709025023862E-233

Elbow Up 5.255741119961993E-279

Right To Left Up 6.042539547765712E-280

Left To Right Up 4.625522678978185E-281

Circle Anticlockwise Up 1.5327924152331677E-299

Circle Clockwise Up 2.2125224996623998E-302

Down Up 1.1530757857757E-310

Up Up 0.0

1-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

27-acc.txt

Elbow Up 1.1361331939778382E-221

Sample Containing Test Results

Square Up 6.928311755261284E-229
Circle Clockwise Up 8.33864480641869E-299
Left To Right Up 7.07856163427E-313
Circle Anticlockwise Up 1.645E-321
Right To Left Up 0.0
Up Up 0.0
Down Up 0.0

17-acc.txt

Elbow Up 2.1263549664398285E-207
Square Up 4.159674567994906E-210
Right To Left Up 4.533054208297709E-221
Circle Clockwise Up 3.555482266718749E-228
Left To Right Up 6.904294218913855E-240
Circle Anticlockwise Up 2.3626869798047272E-240
Down Up 1.166159505915421E-277
Up Up 1.08433689E-315

4/10

0.4

max value 2.1263549664398285E-207

min value 2.6777525623956033E-247

/home/paulo.silva/SAMPLES/test/qnt/perOrientation/screenUp/Test/0/Circle Clockwise Up

14-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

15-acc.txt

Circle Clockwise Up 1.3394E-320
Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Elbow Up 0.0

Sample Containing Test Results

29-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

12-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

36-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

20-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

33-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0

Sample Containing Test Results

Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

30-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

27-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

8-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

1/10

0.1

max value 1.3394E-320

min value 1.3394E-320

/home/paulo.silva/SAMPLES/test/qnt/perOrientation/screenUp/Test/0/Left To Right Up

14-acc.txt

Left To Right Up 9.909649819635956E-183

Sample Containing Test Results

Right To Left Up 2.725776604793838E-220
Square Up 3.106520188096628E-245
Circle Anticlockwise Up 1.0371873421872214E-248
Elbow Up 9.568363148600505E-257
Circle Clockwise Up 1.779097477575242E-262
Down Up 3.5326457486327285E-296
Up Up 0.0

34-acc.txt

Left To Right Up 6.57196E-318
Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

18-acc.txt

Left To Right Up 4.772E-320
Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

31-acc.txt

Left To Right Up 5.008465821347588E-200
Square Up 2.185238872854877E-237
Elbow Up 6.644085780595464E-242
Circle Anticlockwise Up 6.229636604371263E-256
Right To Left Up 2.1859247794583816E-286
Circle Clockwise Up 1.33294125658939E-295
Up Up 0.0
Down Up 0.0

2-acc.txt

Left To Right Up 1.665186087586792E-162
Square Up 8.895970562781675E-208
Elbow Up 3.591315201600554E-239
Circle Clockwise Up 5.588341393657576E-245
Circle Anticlockwise Up 2.7105209897428808E-250

Sample Containing Test Results

Right To Left Up 3.513562024190258E-290

Up Up 0.0

Down Up 0.0

11-acc.txt

Left To Right Up 2.73874366151741E-210

Square Up 4.939336767865348E-225

Elbow Up 1.8544978321199815E-227

Circle Clockwise Up 1.9030313639554113E-241

Circle Anticlockwise Up 3.1794221621547297E-257

Right To Left Up 6.791767577676541E-305

Up Up 0.0

Down Up 0.0

9-acc.txt

Left To Right Up 3.724403826279382E-112

Square Up 5.150080832112866E-155

Right To Left Up 2.474073656990554E-172

Circle Anticlockwise Up 1.5738018779295974E-173

Circle Clockwise Up 6.673251296065141E-197

Elbow Up 9.248352694117201E-200

Down Up 5.95045440936457E-250

Up Up 1.0795578778423319E-283

13-acc.txt

Left To Right Up 3.8864019392110346E-171

Square Up 4.0624945524716053E-202

Circle Anticlockwise Up 7.294885985467277E-223

Elbow Up 8.720635997962219E-238

Circle Clockwise Up 2.185414880960839E-269

Right To Left Up 5.703946628116859E-286

Up Up 0.0

Down Up 0.0

3-acc.txt

Left To Right Up 9.817304105369412E-123

Square Up 1.1582910180841045E-189

Circle Anticlockwise Up 5.317034905747159E-194

Circle Clockwise Up 1.614146896218223E-198

Elbow Up 1.4247756657477076E-218

Right To Left Up 3.273911631580801E-226

Down Up 2.1330166949228346E-259

Up Up 6.3023E-320

1-acc.txt

Sample Containing Test Results

Square Up 8.542286346611228E-138
Elbow Up 9.828574776820243E-166
Left To Right Up 3.4527603833827875E-179
Circle Clockwise Up 3.473715672035625E-181
Circle Anticlockwise Up 3.447468244255546E-195
Down Up 4.955443788897172E-237
Up Up 2.228958032369868E-263
Right To Left Up 8.307510499675279E-308

9/10
0.9
max value 3.724403826279382E-112
min value 4.772E-320

/home/paulo.silva/SAMPLES/test/qnt/perOrientation/screenUp/Test/0/Down Up
14-acc.txt
Down Up 3.808558786749678E-260
Up Up 5.12490777E-316
Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

4-acc.txt
Down Up 2.112394504665011E-258
Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

21-acc.txt
Down Up 1.0483543973111601E-203
Up Up 2.7533909214174734E-276
Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0

Sample Containing Test Results

Elbow Up 0.0

38-acc.txt

Down Up 3.3916369726172603E-171

Up Up 5.53767126540306E-221

Square Up 5.207634378177781E-279

Circle Clockwise Up 7.778794579858374E-302

Left To Right Up 1.6470059231320984E-306

Elbow Up 1.250965086296E-312

Right To Left Up 2.5E-323

Circle Anticlockwise Up 0.0

42-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

33-acc.txt

Down Up 8.343319091509236E-226

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

30-acc.txt

Down Up 2.200452140584909E-38

Up Up 1.6168467072191628E-97

Left To Right Up 4.068106639481639E-159

Square Up 1.4698080190962475E-206

Circle Anticlockwise Up 1.8661793694651344E-212

Right To Left Up 1.266686842787786E-215

Elbow Up 7.744514585571981E-221

Circle Clockwise Up 1.0697916987159763E-233

1-acc.txt

Down Up 5.734466312452765E-219

Up Up 5.969780673E-315

Sample Containing Test Results

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

27-acc.txt

Down Up 2.8340405609704787E-144

Up Up 1.0045693788456049E-180

Square Up 7.726795778637464E-240

Circle Anticlockwise Up 4.065625015480394E-266

Elbow Up 5.3439994723068405E-272

Circle Clockwise Up 5.991197474252296E-277

Left To Right Up 9.821275882006815E-280

Right To Left Up 2.7720324482214E-311

10-acc.txt

Down Up 7.201839877124255E-303

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

9/10

0.9

max value 2.200452140584909E-38

min value 7.201839877124255E-303

/home/paulo.silva/SAMPLES/test/qnt/perOrientation/screenUp/Test/0/Up Up

43-acc.txt

Up Up 3.4352069388940794E-261

Down Up 2.159944957955153E-301

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

31-acc.txt

Sample Containing Test Results

Up Up 1.6945784133930246E-225

Down Up 8.21428243958E-312

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

29-acc.txt

Up Up 1.0E-323

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

7-acc.txt

Up Up 2.428690383560211E-297

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

12-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

9-acc.txt

Down Up 4.96253425287115E-81

Up Up 4.092079728247278E-90

Square Up 3.9672728413492265E-152

Elbow Up 1.3094410006141199E-176

Sample Containing Test Results

Circle Anticlockwise Up 1.2294863849026801E-176

Left To Right Up 1.237983943677692E-181

Circle Clockwise Up 3.473431111876528E-185

Right To Left Up 2.721806277864202E-189

4-acc.txt

Up Up 2.7175775E-316

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

38-acc.txt

Up Up 3.054123924565615E-219

Down Up 2.6633745622992842E-294

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

17-acc.txt

Up Up 6.198601390461482E-97

Down Up 7.646654586208317E-99

Square Up 1.0070173132304763E-180

Circle Clockwise Up 3.0951867583977915E-185

Elbow Up 5.122583477963208E-190

Circle Anticlockwise Up 9.72665856714942E-194

Left To Right Up 2.87401633440514E-208

Right To Left Up 2.5653356476543238E-213

23-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

Sample Containing Test Results

7/10

0.7

max value 6.198601390461482E-97

min value 1.0E-323

/home/paulo.silva/SAMPLES/test/qnt/perOrientation/screenUp/Test/0/Square Up

15-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

5-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

31-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

32-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Sample Containing Test Results

Circle Clockwise Up 0.0

Elbow Up 0.0

22-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

12-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

36-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

3-acc.txt

Circle Anticlockwise Up 0.0

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

1-acc.txt

Circle Anticlockwise Up 0.0

Sample Containing Test Results

Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

35-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

0/10

0.0

max value 4.9E-324

min value 1.7976931348623157E308

/home/paulo.silva/SAMPLES/test/qnt/perOrientation/screenUp/Test/0/Right To Left Up

34-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

6-acc.txt

Right To Left Up 1.6304501140518353E-263
Elbow Up 1.554282978311456E-289
Square Up 4.619806784368467E-301
Circle Anticlockwise Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0

Sample Containing Test Results

16-acc.txt

Square Up 2.291680952344242E-170
Elbow Up 8.092584685886343E-191
Right To Left Up 5.0607433019105994E-197
Circle Anticlockwise Up 1.709865841383426E-198
Left To Right Up 7.260373496861289E-215
Circle Clockwise Up 1.4614855512064272E-220
Up Up 0.0
Down Up 0.0

11-acc.txt

Square Up 4.1768429169174595E-301
Elbow Up 1.2118994762138486E-305
Right To Left Up 3.3285593E-317
Circle Anticlockwise Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0

21-acc.txt

Right To Left Up 1.0580946526617369E-247
Square Up 5.482080845565815E-266
Elbow Up 1.929034092109888E-305
Circle Anticlockwise Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0

20-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

38-acc.txt

Right To Left Up 3.0263464406508134E-196
Square Up 1.3471193380749348E-212
Elbow Up 2.129591823968558E-215

Sample Containing Test Results

Left To Right Up 2.1768080347025414E-217
Circle Clockwise Up 5.14982675341554E-221
Circle Anticlockwise Up 3.5356575012834544E-260
Up Up 2.913309748890851E-295
Down Up 4.9E-324

13-acc.txt

Right To Left Up 5.471398954879195E-239
Square Up 5.408913095078798E-240
Elbow Up 3.919083816800842E-249
Circle Clockwise Up 1.2371691446339813E-252
Left To Right Up 1.5705691726887184E-253
Up Up 1.7585596601327537E-306
Circle Anticlockwise Up 1.45623E-318
Down Up 0.0

1-acc.txt

Square Up 2.1981873068642447E-119
Right To Left Up 1.2190013687042222E-126
Circle Anticlockwise Up 4.7527552832902595E-136
Down Up 8.15945477330943E-149
Circle Clockwise Up 4.032321888484087E-152
Elbow Up 1.4224202620173728E-157
Left To Right Up 3.852894808989167E-171
Up Up 4.117261288107954E-177

23-acc.txt

Right To Left Up 2.400226699965057E-238
Elbow Up 5.1062008810996746E-241
Square Up 2.33987E-318
Circle Anticlockwise Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0

5/10

0.5

max value 3.0263464406508134E-196
min value 1.6304501140518353E-263

/home/paulo.silva/SAMPLES/test/qnt/perOrientation/screenUp/Test/0/Circle Anticlockwise Up

14-acc.txt

Circle Anticlockwise Up 6.11497828746447E-300

Sample Containing Test Results

Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

5-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

31-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

29-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

32-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0

Sample Containing Test Results

Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

22-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

38-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0 Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

33-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

26-acc.txt

Circle Anticlockwise Up 0.0
Right To Left Up 0.0
Square Up 0.0
Up Up 0.0
Down Up 0.0
Left To Right Up 0.0
Circle Clockwise Up 0.0
Elbow Up 0.0

17-acc.txt

Circle Anticlockwise Up 0.0

Sample Containing Test Results

Right To Left Up 0.0

Square Up 0.0

Up Up 0.0

Down Up 0.0

Left To Right Up 0.0

Circle Clockwise Up 0.0

Elbow Up 0.0

10/10

1.0

max value 6.11497828746447E-300

min value 0.0

Sample Containing Test Results

Appendix B

About the Framework Usage

B.1 Gesture Sample File Format

If more than the already trained gestures are to be trained, the files containing the sensor readings should be according to the following rules:

- the files are simple text files which name finishes with the common text file extension: .txt
- each file contains information regarding only one sensor, and a description of the sensor is in the name of the file (*gyro* for gyroscope, *acc* for accelerometer and *lAcc* for the linear acceleration sensor.)
- the file names begin with a number, which identifies the number of the gesture sample (if a sample was generated with two sensors, say gyroscope and the accelerometer, the sample will be composed of two files, for instance 1-acc.txt and 1-gyro.txt, given that the number serves to indicate that those two files are from the same gesture sample.)
- the number and the reference to the sensor from where the values on the file come from must be separated by the "-" character
- the content of the file must only contain the x value, the y value, the z value and the timestamp reported by the respective sensor on each line, separated by the space character and the referred values should be included using the given order. There are no limits to the number of lines inside one of these files.
- samples from one gesture are contained inside a directory with the name of the gesture
- the samples numbering is sequential within a directory and inside each directory, the numbering should begin with 1.

B.2 Functionalities

The software framework developed API makes available to the developer methods with the following headers and functionalities:

- `init(Context appContext, String gestureTemplatesPath)` - initializes the framework configurations, such as registering the sensors used to generate the Hidden Markov Models within the Android application, making the application Context and the path to the directory containing the files describing the models of the gestures available to the framework, and loading the gesture modules from the files to the smartphone's memory;
- `addListener(GestureDetectionListener gdl)` - registers an observer within the gesture detection thread;
- `removeListener(GestureDetectionListener gdl)` - unregisters an observer within the gesture detection thread;
- `start()` - triggers the beginning of execution of the gesture detection thread (the `init(Context appContext, String gestureTemplatesPath)` method only prepares everything the thread needs to be started);
- `stop()` - stops the execution of the gesture recognition thread;
- `registerSensorEventListeners()` - registers the sensors to be used on gesture recognition
- `unregisterSensorEventListeners()` - unregisters the sensors used on gesture recognition
- `detectGesture()` - triggers the gesture detection given the last samples collected
- `resetSensorsData()` - deletes the last readings from the sensors kept in memory