

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# An Approach of a Research Tool based in a Shamanic Interface

Tiago Susano Pinto



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Leonel Caseiro Morgado (*PhD, Habil.*)

Co-Supervisor: António Fernando Vasconcelos Cunha Castro Coelho (*PhD*)

Co-Supervisor: Stephan G. Lukosch (*Dr.*)

June 26, 2015



# **An Approach of a Research Tool based in a Shamanic Interface**

**Tiago Susano Pinto**

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Luís Filipe Pinto de Almeida Teixeira (*PhD*)

External Examiner: Hugo Alexandre Paredes Guedes da Silva (*PhD*)

Supervisor: Leonel Caseiro Morgado (*PhD, Habil.*)

Co-Supervisor: António Fernando Vasconcelos Cunha Castro Coelho (*PhD*)

---

June 26, 2015



# Abstract

Nowadays gesture recognition systems are developed for several purposes, e.g. entertainment, security or education. Such interfaces are controlled by gestures performed by a user, and these gestures are not only associated with a command in the system but also have an intrinsic meaning to the user.

Since gestures can have different meanings for users based on their cultural background, this thesis adopted the perspective of using a shamanic interface. A shamanic interface considers a person's cultural background for gesture-based interaction. From a software architecture perspective, shamanic interfaces use a cultural layer that considers a user's culture when linking a gesture to a certain command. This supports different user experiences with the same interface using gestures linked to a user's culture.

Being a recent proposal, the shamanic interface lacks empirical testing and validation, hence this thesis' contribution is the creation of a research tool to enable empirical research of the concept. After developing a cultural layer, a game was implemented and evaluated to test its performance and applicability. For recognising gestures, the cultural layer relies on a Leap Motion Controller. A Leap Motion Controller can track gestures with a certain precision without requiring the user to wear cumbersome tracking devices. For recognising and classifying gestures, Hidden Markov Models and a respective classifier were used.

By developing a game with a shamanic interface and performing usability tests with users with different cultural backgrounds, it can be concluded that this research tool can be used by empirical researchers to test and develop the concept. Shamanic interfaces are promising, but further work needs to be done on the gesture recognition area as well as in the definition of meaningful cultural gestures.

**Keywords:** Gesture Recognition, Natural Interaction, Human-Computer Interaction, Culture, Shamanic Interface, Hidden Markov Models



# Resumo

Hoje em dia os sistemas de reconhecimento de gestos são desenvolvidos para diversos fins, como entretenimento, segurança ou educação. Estas interfaces são controladas através de gestos realizados por um utilizador, e estes gestos não estão só associados a um comando no sistema, mas também têm um significado intrínseco para o utilizador.

Uma vez que estes gestos têm diferentes significados para os utilizadores com base nas suas culturas, esta tese adota a utilização de uma interface xamânica. A interface xamânica considera a cultura da pessoa para uma interação gestual. De um ponto de vista da arquitectura do software, a interface xamânica usa uma camada cultural que considera a cultura do utilizador quando liga gestos a um certo comando. Isto suporta diferentes utilizadores experienciarem com a mesma interface o uso de gestos ligados a sua cultura.

Sendo uma proposta recente, a interface xamânica carece de testes e validações empíricas, assim esta tese contribui com a criação de uma ferramenta que permite uma pesquisa empírica do conceito. Após o desenvolvimento da camada cultural, foi implementado e avaliado num jogo para testar a sua performance e aplicabilidade. Para o reconhecimento de gestos, a camada cultural depende de um *Leap Motion Controller*. Um *Leap Motion Controller* consegue rastrear gestos com alguma precisão, sem necessitar que o utilizador utilize um dispositivo incómodo. Para reconhecer e classificar gestos, foram usados Modelos Ocultos de Markov e um respectivo classificador.

Ao desenvolver um jogo com uma interface xamânica e realizando alguns testes de usabilidade a utilizadores com diferentes culturas, foi possível concluir que esta ferramenta pode ser usada por investigadores para testar e desenvolver o conceito. Interfaces xamânicas são promissoras, mas ainda é necessário ser realizado um trabalho mais aprofundado na área de reconhecimento gestual como também na definição de gestos culturais significativos.

**Palavras-chave:** Reconhecimento Gestual, Interação Natural, Interação Humano-Computador, Interface Xamânica, Cultura, Modelos de Markov Ocultos





# Acknowledgements

I would like to thank my supervisors, Prof. Leonel Caseiro Morgado, Prof. António Fernando Vasconcelos Cunha Castro Coelho and Dr. Stephan G. Lukosch, for their guidance and all the possibilities created during the short time working in this thesis.

Finally I would like to thank all the people who I had the pleasure to work with, and who helped, inspired and tolerated me during the last five years in this course.

Tiago Susano Pinto



*“Sleep is overrated”*

Unknown



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	4
1.2	Goals . . . . .	4
1.3	Methodology . . . . .	6
1.3.1	The Relevance Cycle . . . . .	7
1.3.2	The Rigor Cycle . . . . .	7
1.3.3	The Design Cycle . . . . .	7
1.4	Thesis outline . . . . .	7
<b>2</b>	<b>Background</b>	<b>9</b>
2.1	Problem Statement . . . . .	9
2.2	Related Work . . . . .	10
2.2.1	Shamanic Interfaces . . . . .	10
2.2.2	Arabic Sign Language Recognition using the Leap Motion Controller . . . . .	11
2.3	Frameworks and Libraries . . . . .	11
2.3.1	Leap Motion SDK . . . . .	11
2.3.2	Accord.NET Framework . . . . .	12
<b>3</b>	<b>Requirements</b>	<b>15</b>
3.1	Typical application based on a SI . . . . .	15
3.2	Requirements . . . . .	16
<b>4</b>	<b>Approach</b>	<b>19</b>
4.1	Gesture detection . . . . .	20
4.2	Creation of models . . . . .	21
4.3	Record lists of sequences . . . . .	22
4.4	Gesture classification . . . . .	23
4.5	Cultural layer . . . . .	24
4.6	Implementation in the game . . . . .	25
<b>5</b>	<b>Implementation</b>	<b>27</b>
<b>6</b>	<b>Evaluation</b>	<b>31</b>
<b>7</b>	<b>Conclusions</b>	<b>33</b>
7.1	Results (Fulfilment of the Goals) . . . . .	33
7.2	Future Work . . . . .	33
	<b>References</b>	<b>35</b>

## CONTENTS

<b>A</b>	<b>Game</b>	<b>37</b>
A.1	Description . . . . .	37
A.2	Menus and Options . . . . .	37
A.2.1	Main Menu . . . . .	37
A.2.2	Settings . . . . .	37
A.2.3	Game . . . . .	37
A.3	List of Actions . . . . .	38
<b>B</b>	<b>Cultural Gesture Analysis</b>	<b>39</b>
<b>C</b>	<b>Detailed Implementation</b>	<b>43</b>
C.1	ShamanicInterface.Culture . . . . .	43
C.1.1	CulturalPair . . . . .	43
C.1.2	CulturalLayer . . . . .	43
C.2	ShamanicInterface.DataStructure . . . . .	45
C.2.1	Sign . . . . .	45
C.2.2	Sequence . . . . .	45
C.2.3	SequenceList . . . . .	46
C.2.4	SequenceBuffer . . . . .	47
C.3	ShamanicInterface.Classifier . . . . .	47
C.3.1	HMMClassifier . . . . .	47
C.4	ShamanicInterface.State . . . . .	48
C.4.1	Actions . . . . .	48
C.5	ShamanicInterface.Utils . . . . .	48
C.5.1	Transform Leap Motion Data into SI Data . . . . .	48
C.5.2	Save and Load Frames and HMMs . . . . .	49
C.5.3	Creation of gesture models . . . . .	50
C.5.4	Acquirement of gestures' models . . . . .	50
C.6	Recorder . . . . .	51
C.6.1	Program . . . . .	51
C.6.2	Recorder . . . . .	52
C.6.3	RecordListener . . . . .	52
C.7	Application - Maze Game . . . . .	53

# List of Figures

1.1	20 emblem gestures [MMC80]	5
1.2	Design Science Research Cycles [Hev07]	6
2.1	Leap Motion	11
2.2	Leap Motion Gestures	12
3.1	Example of game gestures	15
3.2	Example of objects in the game	16
3.3	An initial approach on the application's architecture	16
4.1	Representation of all features selected	20
4.2	Example of the initial settings of a gesture being recorded	22
4.3	Control flow inside the system when creating and using a classifier	25
5.1	Screenshots of Prototype 0	27
5.2	Recording the gesture POINT FRONT	28
6.1	Example a user performing a gesture	31
6.2	Moving Forward gesture	32

## LIST OF FIGURES



# List of Tables

1.1	List of emblems and their meanings and cultural backgrounds [McN92] . . . . .	3
4.1	Result of the small analysis in both cultural backgrounds . . . . .	19
B.1	Response gestures . . . . .	41

## LIST OF TABLES

# Abbreviations

<b>API</b>	Application Program Interface
<b>GI</b>	Gesture Interface
<b>HMM</b>	Hidden Markov Models
<b>NUI</b>	Natural User Interface
<b>SDK</b>	Software Development Kit
<b>SI</b>	Shamanic Interface
<b>UI</b>	User Interface
<b>Unity</b>	Unity 3D (game engine)
<b>VR</b>	Virtual Reality



# Chapter 1

## Introduction

Gestures made by humans in their daily life are filled with information. Information that is available to be used for several systems in various areas of study. By only gathering, analysing and presenting data, or even by assign the gesture as a command in the system, allows, with this, the user to interact with it. Gesture recognition interfaces are being developed to offer a solution to this contemporary kind of Natural Interaction.

Some examples where these gesture recognition interface were already implement are:

- Medical supervising (through the analysis of emotion or stress levels); [[KSG<sup>+</sup>13](#), [MA07](#)]
- Security systems (emotion's analysis, lie detection, drivers' surveillance); [[BS12](#)]
- Serious games; [[HPKJ12](#)]
- Sign language; [[MAD14](#)]
- Remote control of drones; [[TFST14](#)]
- Aid in the communication between children and computers; [[MA07](#)]
- Interaction in virtual environments; [[MA07](#)]

To respond to this wide set of application, there are a huge number of devices for acquiring movements. They can detect the user's gestures through different stimulus (electric, optic, acoustic, magnetic or mechanic) [[BS12](#)]. Some examples of these devices are:

- **Digital gloves** allow to capture the hands' position and measure the finger joint angles, but the user is encumbered with the device.;
- **Devices based on computer vision** have the advantage of allowing the user to preform the gestures without having a device attached with the body, but they suffer from temporal and spatial segmentation and they are influenced by the environment around the user [[WXX<sup>+</sup>09](#)] (*e.g.* Video cameras , *Leap Motion*, *Microsoft Kinect*);

## Introduction

- **Accelerometers**, also common with the digital gloves, require that the user hold them while capturing the movement, but are extremely useful for the detection of large gestures (*e.g. Nintendo Wii, Sony Move*);
- **Electromyographic sensors** are useful in the detection of wrist movements and fingers, by reading the electric impulses created by the contraction in the arm's muscle of the user. But, due to problems in the measure of them, the number of different gestures that might be detect are reduced compared with other devices [WXK<sup>+</sup>09] (*e.g. Myo*);
- **Magnetic markers** grant the detection of the position and orientation of each marker, but they are affected by variation in the magnetic field and have a fixed maximum number of markers that might be detected (*e.g. Liberty Latus*);
- **Ultrasonic sensors** detect the position and the velocity of the gestures, but are limited in precision and response time, because the sound proprieties (*e.g. Mimesign*).

The specification of these devices changes in different aspects, such as: range, resolution, linearity, response time, cost, size, weight, environmental robustness and operation life [BS12]. Different devices have different ways to track and acquire the user's gesture. An example in computer vision devices is the use of algorithms to detect and track the hand of a user, with the detection of his/her skin colour and the hand and finger shapes with adaptive algorithms and the analyses of three dimensional histograms [SWTO04].

To classify the gestures, after they being captured and modelling, there is several techniques that are already developed, based on the same approaches to support other areas with similar challenges. Some of these algorithms use artificial neural networks and state machines (e.g. HMM [MA09], FSM [IK12], DBN [AMSP14]), others create clusters (K-mean [AMSP14], Fuzzy Clustering Algorithm [IK12]), or by using others algorithms related to artificial intelligence such as genetic algorithms[IK12]. A great part of the methods already developed use a stochastic concept, algorithms based on probabilistic methods, because it creates designs that are more robust and error tolerant.

Looking into the nature behind gestures produced by society, it is possible to separate them in 5 distinct types: [MA07] gesticulation, movements that follows a person's speech naturally; language like gestures, gestures that replace words or phrases in a speech; pantomimes, gestures that describe objects or actions; emblems, cultural signs (such as 'V' of victory); sign languages, gestures associated with an well defined linguistic system.

The emblems have a strong link to the culture background that support it, and therefore there is distinct meanings for the same gesture when performed in different cultures. Some examples of this cultural gestures are presented in the following table (Table 1.1) with the representation of each gesture in the Figure 1.1

## Introduction

Table 1.1: List of emblems and their meanings and cultural backgrounds [McN92]

<b>Sign Name</b>	<b>Cultural background</b>	<b>Meaning</b>
Fingertip Kiss	Holland, Belgium, Yugoslavia and Turkey	Praise
	Portugal, Sardinia, Malta and Corfu	Salutation
Finger Cross	England, Scandinavia, parts of Sicily, and Yugoslavia	Protection
	Corfu and Turkey	Breaking a friendship
The Nose Thumb	Everywhere	Mockery
The Hand Purse	Italy, Sardinia and Sicily	Query
	Portugal, Greece, and Turkey	Good
	Belgium and France	Fear
	Holland and Germany	Emphasis
Cheek Screw	Italy, Sicily and Sardinia	Good to eat
	Germany	Crazy (if made on the side of the head)
Eyelid Pull	Italy	Watch out, be alert
	Everywhere else	I'm alert
Forearm Jerk	North America	Italian Salute
Flat-Hand Flick	Belgium, France, Italy and Greece	Beat it
Ring		OK
	Tunisia	Threat
	Brazil	Insult
	Germany, north Italy, northern Sardinia, and Malta	Orifice
	Belgium, France, and Tunisia	Zero
Vertical Horn	Spain, Portugal and Italy	Cuckold
Horizontal Horn	Malta and Italy	Protection
Fig		Sexual comment
		Insult
Head Toss		Negation
		Beckoning
Chin Flick		Disinterest
		Negation
Cheek Stroke		Thin and ill
Thumb Up		OK

*Continued on next page*

Table 1.1 – *Continued from previous page*

Sign Name	Cultural background	Meaning
Teeth-Flick		Nothing
		Anger
Ear Touch		Effeminate
		Watch out
Nose Tap		Secrecy
		Insult
Palm-back V-sign		Victory
	Britain	Sexual insult

## 1.1 Motivation

It is highly desirable for a user to experience a natural interaction with a computer when working with a Natural User Interface, such as a Gesture Interface. So, it is expected that this interface provides an intuitive and unambiguous interaction to the user.

Based on the globalization that the world is currently going through, we may notice that diverse number of application developed with this kind of interfaces reaches a wide population with a most diversity kind of cultures. Therefore, in order to maintain the intuitive and unambiguous features expected in NUI's, these interfaces must consider the differences between the distinct cultures.

In the gestures, some of them (emblems), there is also differences between cultures. We can notice that user from a specific cultural background has different gestures from someone else with another cultural background. And once these gestures are connected to the perception of performing a certain command in a gesture interface, the user's cultural background is a factor to consider in an interface of this kind.

## 1.2 Goals

This thesis aims to develop and utilise a shamanic interface in a research tool to enable empirical research of the concept. To this end, the SI use different sets of cultural gestures (emblems) and the cultures associated to each one of them. The users will be able to select, based on their culture, the gestures of their own cultural background to command an application.

Given an example, a computer game and two different cultures, it is crucial to gather all the application's commands as well the gestures that represent those actions in distinct cultures. It is also needed to understand how the gesture is going to be captured, modelled, and later recognized. Therefore, this project focuses on expanding a shamanic interface by applying the architecture



Introduction



STEIN AND DAY/Publishers/Scarborough House,  
Briarcliff Manor, N.Y. 10510

Figure 1.1: 20 emblem gestures [MMC80]

behind and implementing in the application, by explaining the steps needed to use it in the empirical tool. This is to demonstrate how to operate it, test it and improve it, based on the adversities expected in a real case study, resulting in a system that can be used by empirical researchers to test and develop more the concept.

### 1.3 Methodology

This thesis follows a design science methodology. This procedure has its roots in engineering, what Herbet Simon described as "*the sciences of the artificial*", and focuses on creating and evaluating innovative artefacts. These artefacts, and subsequently the design science paradigm, are built to deal with unsolved problems. Since the known theory is often insufficient, it relies not only on a knowledge base, but also on the experience, creativity, intuition and general problem solving methods of the researcher. He (or she) is also responsible for evaluating the artefact and analysing the feedback received from it, in order to understand the utility that it provides to solve the initial problem and improve the research' knowledge about it. This loop, since the starting point, the creation/development of the artefact and the evaluation, repeats through the all design science process, with the purpose to improve the environment, the artefact and the knowledge base[HM03].

Therefore, it is possible to notice three distinct cycles inside the design science research paradigm [Hev07], as showed on figure 1.2. The Relevance Cycle, that is based on the contextual environment creates and test the requirements. The Rigor Cycle, that links the current knowledge base situation with the activities. And the Design Cycle, where the main iterations between the building and evaluation of artefacts occur. These three circles are autonomous and should occur in parallel. The results from the in-between cycles and, above all, the end phase in the design cycle contribute to the enlargement of the knowledge base.

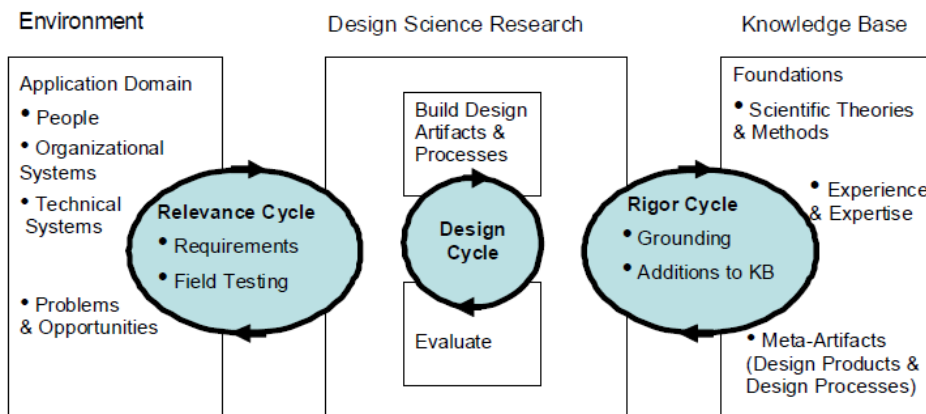


Figure 1.2: Design Science Research Cycles [Hev07]

### 1.3.1 The Relevance Cycle

This first circle in the methodology is called *Relevance Cycle*, and it is where the design science start. Based on the motivation and in the opportunities to improve the environment, this cycle defines the criteria and requirements that the developed activities must present to enhance the initial environment. From the creation of new artefacts, the results from the field tests may show that these have defects on the initial requirements were incorrect to solve the problems presented by the original background. Therefore, a new iteration of this cycle must be performed, so it can produce both as new requirements as well as final criteria that the created artefacts must take into consideration.

### 1.3.2 The Rigor Cycle

By supporting the work with the past knowledge of scientific theories, engineering methods and existing artefacts, the *Rigor Cycle* creates the foundations where the design science research takes place. This provides not only the tools to develop new artefacts, but also ensures the innovation of the current knowledge base. When developing the design science research, additional grounding knowledge might be necessary to support and improve the created artefacts, resulting in a new iteration in the rigor cycle. In the end of each cycle, the results of the activities developed in the methodology, might originate new theories, methods or tools as additions to the original knowledge base.

### 1.3.3 The Design Cycle

This last cycle of science design, labelled as *Design Cycle*, is the core of any design science research project. This cycle, normally, iterates more often than any of the other two cycles, since it is where the new artefacts are created, developed and tested several times without the need for iterations in the other cycles. While the developed of the artefacts is based on the requirements from the relevance cycle and the grounding knowledge base from the rigor cycle as an initial inputs in this cycle, the feedback from the evaluation is the main transitional addition between design cycles. Thereafter, these evaluations should be initially made on a controlled experimental situation before being field tested, simultaneously with the relevance cycle. This creates multiple iterations on the design cycle and, by balancing the construction and the evaluation parts in this cycle, is it possible to achieve an ideal and sustainable use of efforts in this cycle on the design science research.

## 1.4 Thesis outline

The rest of the thesis adopt the following structure:

Chapter 2 introduces the problem, by explaining the concept in which the project is based, and what is meant to achieve with this work. Along with it, the background knowledge used in this thesis is also presented, being therefore related to the Rigor Cycle in the adopted methodology.

## Introduction

Chapter 3 demonstrate the work based on the relevance cycle, presenting the requirements and specifications needed to accomplish the creation and use of tools for this project.

Chapter 4 describes the approach used to develop and implement the tools needed in this work.

Chapter 5 details the implementation of the approach through progressive versions of the project in the design cycle.

Chapter 6 describes the final evaluation performed in order to gather feedback from users through the usability tests.

Chapter 7 draws conclusions over the developed work and suggests possible directions for future projects.

## Chapter 2

# Background

In this chapter the problem statement is presented, as well as the frameworks and libraries involved in the development of the work, and related works of others authors.

### 2.1 Problem Statement

Ever since the interaction between users and computers was developed, the Human-Computer Interaction field has been evolving. The Graphical User Interface (GUI) paradigm, is still the main User Interface (UI) used by the masses, but this paradigm is being replaced by a new pattern of UI, the Natural User Interface (NUI). As we can observe in several studies and improvements in this area, there have been new approaches of communication between humans and computers, in order to provide a more natural interaction between them.[\[dC14\]](#) The technological and scientific advances in this area, more specific, in the gesture recognition interfaces, are noticeable in the development of new devices and algorithms to capture and process the position and motion of a user's body, normally with a special interest in the extremities (*e.g. arms, hands and legs*).

But most of the gesture based NUIs developed nowadays have some sort of limitation related to the performance of the commands. Consequently, they normally adopt a mimicking approach. With this, the user just follows along and tries to imitate the gestures shown to him to confirm an option or to evaluate its performance (such as in dance games). With this, the system only needs to calculate the similarity between a player's motion and the intended ones. This has some issues and limitations, such as difficulty of learning or performing some gestures, as well as the incapability to be used by an handicapped person on a physically level.

Although, by figuring out a gesture associated with a certain action with a more relevant meaning, it is possible to go over the simple mimicry approach to command applications. This is usually achieved by determining the analogue gesture that represents the instruction intended. It may be considered a solution, where users might define their own set of gestures, but the potential to escalate its complexity is enormous when one considers the endless number of gestures that a user

## Background

might try in order to achieve the diverse commands held by the environment in cause[Mor14]. The American novelist Daniel Suarez presented a solution for this problem when he wrote about a *shamanic interface* in the science fiction novel "*Freedom<sup>TM</sup>*":

*"It's called the shamanic interface because it was designed to be comprehensible to all people on earth, regardless of technological level or cultural background."*  
[Sua10]

Even though this idea seems to point to a gesture interface where a single set of gestures is intrinsically understandable by all kind of people, due to cultural differences in those, this utopian interface is impossible in such cultural diversity in the current world. On the other hand, it can be considered a shamanic interface (or anti-shamanic interface) were multiple set of gestures, depended on the user's cultural background, can control the same system. Therefore, this interface can exceed the mimicry approach, by embracing the features associated with such in-born characteristics of cultural gestures, in usability, learning and memorizing, and accessibility aspects.[Mor14]

To achieve this kind of interface a cultural layer must be developed on the software designed for a gesture recognition interface. This layer it will be responsible for mapping gestures performed by the user to the command in the application, based on the user's culture. This results in a interface that is capable of recognizing the gestures attached to a certain command in the user's cultural beliefs. This can be accomplished by creating a lookup table between the cultural gesture and the wanted action. [dC14]

The creation of a computer application as a case study enables the testing and enhancement of the previously referred interface's architecture. Therefore, this thesis implements and uses such interface over a computer game, in order to develop a shamanic interface capable of standing up to a pratical case study, by improving it to overcome the challenges presented by the application where it is used.

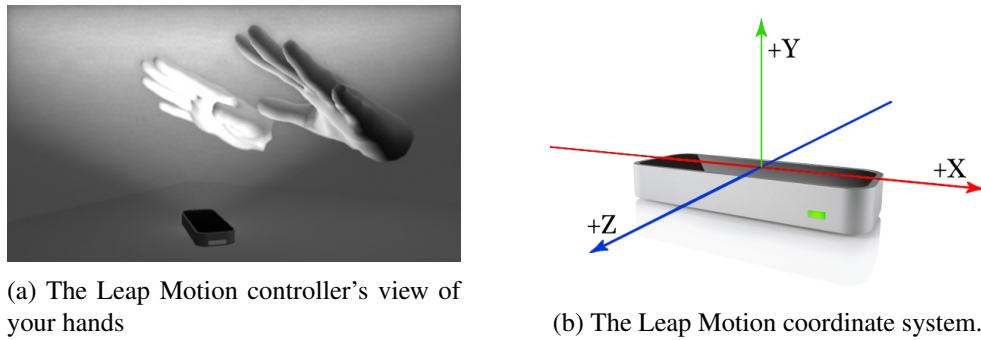
## 2.2 Related Work

### 2.2.1 Shamanic Interfaces

Being a recent concept, the only approach found on a SI it was the previous work [dC14] that this thesis follows. The end application consisted in a scene where was possible to observe the movement of a cube based on gestures performed and the culture selected in a separated interface. Being a proof of concept, the interface developed uses as cultural gestures, swipe movement patterns recognized through an algorithm over the collected data in Microsoft Kinect, and abstract cultures to represent the different user's cultural backgrounds. The shamanic interface, after recognize the gesture performed, links it to the respective feedback, with a lookup table, where given a pair of a gesture's name and a culture returns the command on the application. In this case the consequences are based on keystrokes that are used on the end application.



## Background



(a) The Leap Motion controller's view of your hands

(b) The Leap Motion coordinate system.

Figure 2.1: Leap Motion  
[Mot15a]

### 2.2.2 Arabic Sign Language Recognition using the Leap Motion Controller

The project [MAD14] uses a Leap Motion Controller to recognize 28 letters in the Arabic Sign Language. It tests the performance of two classifiers, the Naive Bayes Classifier and the Multilayer Perceptron, in 12 features extracted from the Leap Motion controller (*i.e.* finger length, finger width, average finger tip position in a x, y, and z-axis, hand sphere radius, palm position in a x, y, and z-axis and palm direction with pitch, roll and yaw).

## 2.3 Frameworks and Libraries

### 2.3.1 Leap Motion SDK

The Leap Motion controller is a capture device developed by *Leap Motion Inc.* to track the position of objects like hands, fingers and finger-like (*e.g.* pen tips) objects in a three-dimensional space (Figure 2.1b). It uses two IR cameras and three LEDs to capture the environment, but a more detailed information of how it creates a 3D scene was not yet revealed by *Leap Motion Inc.*. The controller can track objects in the upper part of the device, Figure 2.1a, with an approximated field of view of  $150^\circ$ , a capture range between 2.5 cm and 1 meter, a framerate between 50 and 200 fps and an error of less than 2.5 mm. [MAD14, WBRF13, Mot15b, NPWZ14]

All access to the data tracked by the device is made by the API provided in Leap Motion SDK. A representation of a portrait with all tracked objects can be obtained in a `Frame` object, being considered for that the main base of all Leap Motion data model. In `Frame` structure can be found a list of hands, the main entity tracked by the Leap Motion controller, presented in the instantaneous snapshot of the scene recorded, tracked through the validation between the data from the sensors and an inner model of the human hand. The frames access can be performed through two ways: by a polling method in the `Controller`, that access to the most recent `Frame` or a previous one that is stored in the small history buffer, or with a callback method, by creating a `Listener` object as a handler in the `Controller` to deal with several events, such as when a new frame is available. The former method offers the best strategy when the application already has a natural frame rate,

## Background

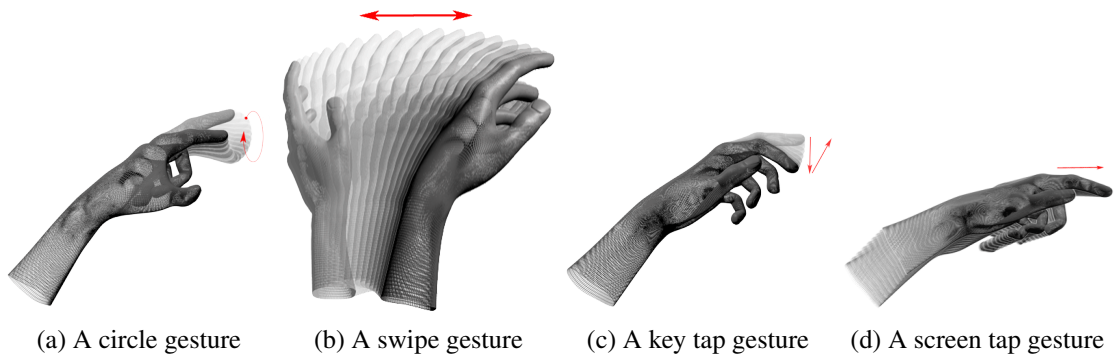


Figure 2.2: Leap Motion Gestures  
[Mot15a]

such as the developed one, while the latter one implies the handle of multiple threads, creating a more complex system.

Each `Hand` incorporates the physical characteristic represented in a human hand, such as the hand position, direction, movement and also a list with all five fingers, it is assigned an unique ID value to each hand while it is visible within the field of view in consecutive frames. When removed from the field of view and re-inserted later a new ID is assigned.

The fingers, consecutively, represent the five finger on the typical human hand (in this abstract context the thumb is also consider a finger), and each one of them contain many attributes about all the aspects in it, as the bones, joints, positions and directions. This thesis uses the direction and position of each finger-tip.

It is also possible to obtain a list of certain movement patterns, like circles (Fig. 2.2a), swipes (Fig. 2.2b), key taps (Fig. 2.2c) and screen taps (Fig. 2.2d), presented in each frame produced during the gesture's lifetime. The recognition of each `Gesture` is disabled by default, but can be enabled in the `Controller` class, as well as other configuration parameters that are connected to them can be customized, such as minimum distance, length, velocity or radius, depending in the gesture type, that represent the minimum criteria for a gesture to be recognized.

### 2.3.2 Accord.NET Framework

The `Accord.NET` Framework provides methods and algorithms through several different libraries on diverse scientific computing techniques of computer vision and artificial intelligence, such as machine learning, mathematics, statistics, computer vision and computer audition, by extending the `AForge.NET` Framework, another framework in this field of expertise. [dS12]

In this framework it is possible to create Hidden Markov Models with a specific topology, with the initial state probabilities and the matrix of transition probabilities, and emission density, with the probability distributions for each state. After the construction of a HMM with arbitrary-density state probabilities, it can be trained with a supervised or unsupervised learning algorithm. While a supervised learning algorithm expects the observed sequence and the sequences of states in the HMM, an unsupervised learning algorithm only expects the former sequence. [dS14]



## Background

Hence, based on a list of models previously created, through a classifier is possible to recognize the most likelihood model to produce a given sequence.

## Background

## Chapter 3

# Requirements

### 3.1 Typical application based on a SI

An application that uses a shamanic interface, must be an application where users with different cultures perform meaningful gestures for commands. Therefore, the application must be characterized not only by the use of gestures to control it, but also by the selection of commands that are possible to perform with a meaningful gesture for each cultural background.

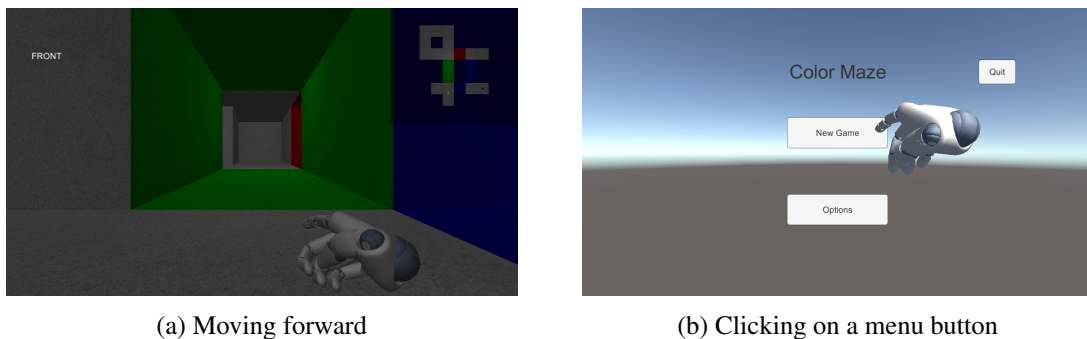


Figure 3.1: Example of game gestures

An example of an application of this kind is a maze game, where the player must perform certain actions in order to reach the labyrinth's end. The actions selected to be executed by the character in the maze, should be more than simply moving (Figure 3.1a) or interactions with the environment (Figure 3.2) (such as walking forward/backward, rotating to the left/right or grabbing a cube), but also actions with a certain level of abstraction to the user, such as mute/unmute or pause/resume. Trying therefore to achieve a full spectrum of possible cultural and meaningful gestures and in turn a higher probability of finding differences in the gestures. Even in menus, inside or outside the game level, when it is not possible to create meaningful gesture to control them, the application must offer other set of gestures such as tapping a button (Figure 3.1b), mimicking the tap of a keyboard.

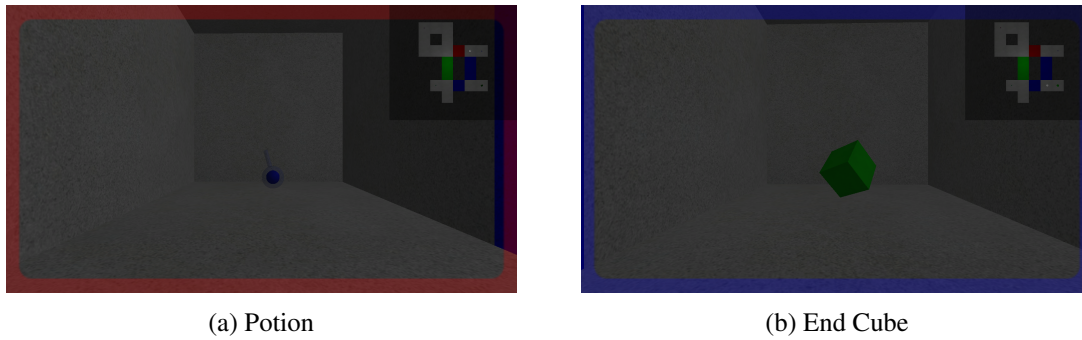


Figure 3.2: Example of objects in the game

### 3.2 Requirements

Based on the previous example of an application that uses a shamanic interface and an initial architecture (Figure 3.3), the research tool developed assembles the following requirements:

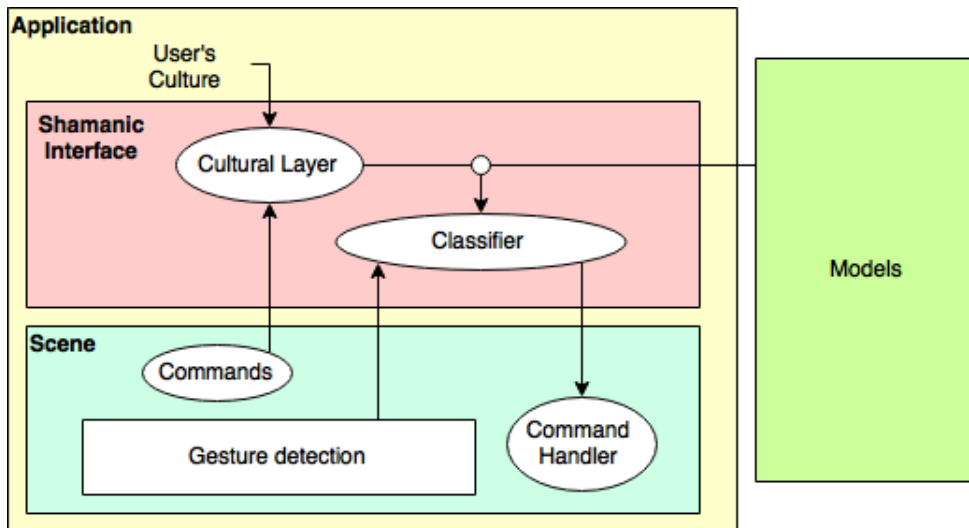


Figure 3.3: An initial approach on the application's architecture

#### R1. The Cultural Layer must return a list of gestures to be recognized

Based on the list of commands to be recognized and in the user's culture, the shamanic interface must create a list with all the gestures' names in the current cultural background to detect all the commands needed.

#### R2. The Cultural Layer must be independent from the model type used to define gestures

To turn the Shamanic Interface adaptable to other gestures models, the gestures definition in the cultural layer must be autonomous from the selection made to modelling the gestures.

**R3. The Classifier must only classify a selected set of gestures**

By reducing the number of gestures classified, it increases the classifier's performance, not only by reducing the number of gestures that are evaluated, but also by reducing the number of possible models to be wrongly selected. This implies that the application must create sets of commands used in each scene.

**R4. The Classifier must return the command associated to the recognized gesture**

After receiving and correctly recognizing a sequence of gestures, the classifier must retrieve the name of the command associated to it.

**R5. The Application must offer two or more cultural option to the user**

In order to demonstrate the different sets of cultural gestures for each kind of cultural background, the application must allow an user to select between more than 1 kind of culture.

**R6. The Application must offer an option for people that are not covered by the cultural selection available**

Since the end application may not have a cultural option for all users, a default group of gestures must be available in the application to offer a option for those cases.

## Requirements

## Chapter 4

# Approach

On a very initial phase, by targeting two different cultures such as the Portuguese culture and the Dutch one, it was found, based on documentation, differences in cultural gestures between the European regions which both cultures belong, respectively south and middle of Europe.

Table 4.1: Result of the small analysis in both cultural backgrounds

Command	Cultural background	Gesture
Move Forward	Portuguese	POINT FRONT
Move Forward	Dutch	OPEN HAND FRONT
Move Backward		POINT BACK
Rotate Left	Portuguese	POINT LEFT
Rotate Left	Dutch	OPEN HAND LEFT
Rotate Right	Portuguese	POINT RIGHT
Rotate Right	Dutch	OPEN HAND RIGHT
Grab		GRAB
Drink	Portuguese	MIMIC BOTTLE
Drink	Dutch	MIMIC GLASS
Pause		HALT
Resume	Portuguese	INDEX ROTATING
Resume	Dutch	HAND ROTATING
Mute		INDEX
Unmute		MIMIC MOUTH
Quit		WAVE
Number 1		INDEX
Number 2		INDEX+MIDDLE
Number 3		INDEX+MIDDLE+RING
Yes		THUMBS UP
No	Portuguese	THUMBS DOWN
No	Dutch	WAVE NO

With those differences in mind, it was possible to create an application's idea that took advantages of the gestures' characteristics contrast between both cultures. The idea behind the application was to create a small game where a user has to travel through a maze and perform certain

## Approach

actions to be able to reach the end and, in consequence, complete the game level. In order to gather all the gestures both cultures use to interact and control the system, it was performed a small analysis between the gestures performed by two persons with the two different cultural backgrounds that were targeted initially when asked to achieve a certain command through a meaningful gesture. The results of this analysis are presented in Table 4.1. It was hence possible to create a list with the 22 gestures, from two different cultures needed, to be recognized. The requirement R5 is therefore achieved.

In order to develop research tools based on a shamanic interface and implement them in the final application, this work addresses the following approaches in each design concept presented below.

### 4.1 Gesture detection

The gesture detection is made through the Leap Motion Controller, a capture device capable of detecting and capturing the hands of the users. In the game, the use of the Leap Motion Controller was implemented with the help of Leap Motion SDK. The API in it allows the developer access to a `Frame` structure that can be polled from the `Controller`. It is the `Frame` that keeps the tracking data of a capture set of hands. Each hand has all the physical characteristics of the user's hand, such as the palm position, velocity and direction, and all the five fingers, also with their respective position, velocity and direction.

Based on this, and since the classifier and the models work with sequences of sets of values, it is possible to create a structure that stores all the desired values into a single set, `Sign`, and a sequence of them, `Sequence`. Thereafter, it was also developed a function capable of turning the `Hand` from the Leap Motion SDK into the `Sign`. The selected features were the directions of all five fingers and the hand's palm, creating a single array of floating-point values, as represented in Figure 4.1.

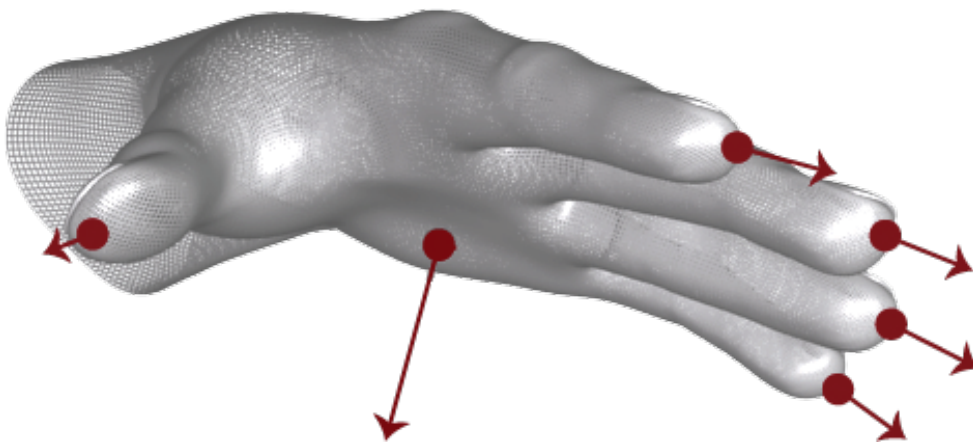


Figure 4.1: Representation of all features selected  
[Mot15a]



## Approach

While all the sets in a single sequence and between each sequence must have the same dimensions, the dimension of each sequence may alter, and it is this variation of the number of signs inside of each sequence that differ a static gesture from a dynamic gesture. Although a static gesture has a small sequence with tenuous or non-existent fluctuation in the set of values, and a dynamic gesture is represented by a longer sequence with variations between each set, the recognized sequence always had a static and large range in order to detect dynamic gestures. Therefore, it was not considered the variation of the hand's position, because it miss recognized a static gesture for any dynamic gesture when the former was executed with a certain hand movement, inherent to the human hand, even while resting, possesses a certain amount of residual movement. It was also detected that, since all the dynamic gestures selected had variations in the direction of the fingers and/or in the hand's palm during the performance, it is possible to distinguish them only with the 6 directions initially refer, 18 values, as each direction is represented by a three-dimensional vector with its 3 components (x, y and z).

## 4.2 Creation of models

The creation of hand gestures models is sustainable by establish connection between the gesture designation, its name, and the corresponding Hidden Markov Models. The HMM structure and methods used is available through the Accord.NET Framework, that allows the developer to create, teach, store and load the model.

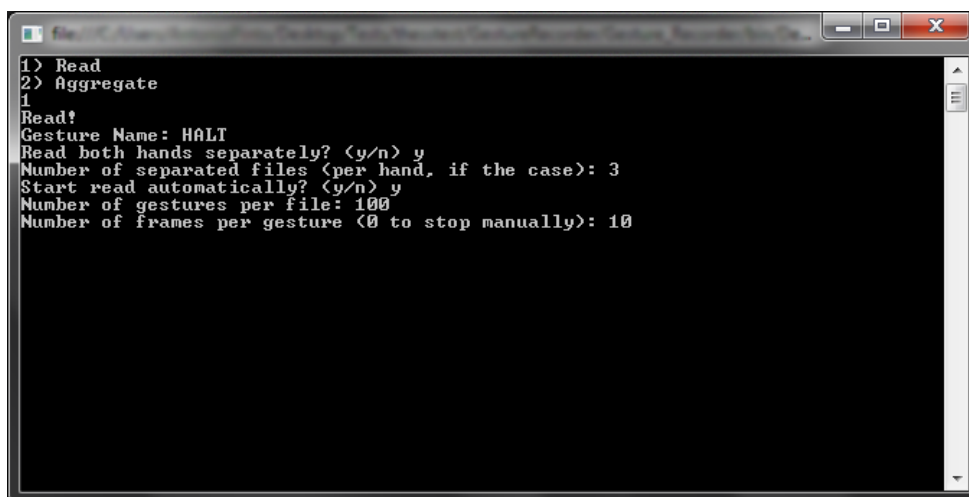
The model is initially started based on a forwarding topology, a state-transition topology where the current state cannot go back to a previous one, and a multivariate Gaussian distribution with the dimensions of the set size in the sequences that will train and be evaluated in the classifier afterwards. Hence, it is created an instance of the Baum-Welch learning algorithm that runs over the HMM created previously with the learning sequences, teaching and tuning the parameters of the model to recognize the gestures and returning the likelihood of a given sequence be produced by the model. The learning algorithm created has some stop parameter during the teaching process, such as a tolerance value that will stop the procedure when the changes in the likelihood for two consecutive iterations is lower than the parameter.

These learning sequences are created from a recorded file that contains a set of sets of frames with the hand gestures. While a set of frames represent a single sequence of hand signs through time that represents the full gesture, the list of sets represent the several examples of the same gesture used to train the model in order to increase the trustworthiness of the gesture created. The structure *SequenceList* was created to support the creation of models, by storing a set of sets of frames after being transformed into a list of sequences. Finally, the model is saved, so it can be loaded when the game start, without the need to recreate all models.

### 4.3 Record lists of sequences

To support the creation of models, a recorder of gestures was developed. This recorder uses the Leap Motion API to create a `Listener` of frames that is added to the `Controller`. The created `Listener` overrides the callback function of whenever a new frame of a hand is available in order to aggregate and store them. By supervising the listener created with the recorder it is possible to manage when the recording of frames start and when its ends. By storing multiples sets of frames recorder, it is created a file with the same gesture recorder numerous times with slight differences between them, in order to trying to represent all the various ways to perform a gesture, since the values (physic characteristics) that represent gesture and the velocity that it is performed changes between person to person and even between the same user between different executions. To turn the recorder independent of the hand features, selected values from the `Hand` structure, used in the models, the recorder stores set of frames and not a *Sequence* (a sequence of `Sign`'s objects), so the selected components may change later, without the need of recording the same gesture again.

Therefore the developed program can record several sequences of frames into the same file, or diverse files, with several sequences of frames. Since sometimes the device misreads the correct position or trajectory of the hand, by allowing to record all the sequences of the same gesture through different files, tolerates errors that might occur when recording the gestures. With this, if a small error occurs when recording the gesture, only the file that contains the error must be removed and replaced by a new one, preventing a small error to ruin all the samples stored in a single file and the need of record all the sequences again. When recording it is possible to manually start each sequence individually (after the end of the previous one), or automatically. The later option is achieved by determining that the following sequences can start right after the end of the previous. It is also possible to determine a fixed number of frames to be stored in each sequence or let the user to signal its terminus.



```

1) Read
2) Aggregate
1
Read!
Gesture Name: HALT
Read both hands separately? (y/n) y
Number of separated files (per hand, if the case): 3
Start read automatically? (y/n) y
Number of gestures per file: 100
Number of frames per gesture (0 to stop manually): 10

```

Figure 4.2: Example of the initial settings of a gesture being recorded

With this, when start reading the gesture, the research must indicate the main name of the

## Approach

file, the number of files, the total of sequences in each file, the number of frames recorder in each sequence (zero to indicate the end of each sequence during the record), and if it wants that the register of sequence start automatically or not (Figure 4.2). It is also possible to select if the files should have the consideration that both hands are going to be recorded separately with the same file name, doubling the number of files created, or not. The end name of each sample file is composed by the file name indicated by the user, "R"/"L"/"" to indicate which hand was recorded, and the file number (starting in 0). The files are stored in a sub folder Sample to help managing the created files.

In the end, after recording the gestures without errors in different files the user can aggregate all the files correspondent to the same gestures. The aggregation join all the sets of sets of frames into a single set of sets. In case that both hands were recorded individually and because of the stop parameters' nature in the models' learning algorithm, the sequence of both files are joint alternately, in order to create a single file with sequences of both hands by turns.

### 4.4 Gesture classification

The classifier module is responsible for computing the most likely model for the given *Sequence*, a list of set of values that represent the full gesture, by using the class for Hidden Markov Model sequence classifier available in the Accord.NET Framework. A new instance of a classifier is initialized in the beginning of each state in the game in order to analyse, when prompted, the likelihood of a sequence in its models. The classifier is initialized not only with the models, but also the names associated to them. By associating a name to a model, when the classifier computes the most likely probably of fitting a sequence, it is capable of returning the associated name, instead of the model. The number of models with which the classifier is charged affects its performance. The increase of a new model, creates an error probability associated with the accuracy to determine de correct model in the presence of another one. Therefore, given the intrinsic error associated with gesture recognition, only the gestures needed should be analysed, since a vast number of models needed to classify will increase the possibility of a misclassification. With this, both requirements R3 and R4 are fulfilled.

In order to assist the classification, a special sequence with a fixed dimension was developed, that when it is full and a new sign is added, the first and oldest element on it is removed. This works like a buffer that keeps the last maximum value elements in the sequence. Since the gestures are continuously read, there is not a way to determine the starting and end point of a sequence of signs performed by the user. Therefore, with this buffer it is possible to achieve the last sequences with a given capacity and retrieve it to the classifier, by continuous creating the most recent sequence available with a maximum size. This will always represent the full gesture after the user has performed it, even if the gesture performed uses fewer frames that the buffer's maximum size.

## 4.5 Cultural layer

While in a previous work[dC14] it was proposed an approach of a map from a pair of a gesture name and the respective culture to a command name, to retrieve the correct consequence in the current system, based on a gesture's classification approach, a reverse concept, where a pair between the command and the user's culture is linked to a gesture name, seemed to be the most suitable.

In order to take advantage of a classifier's perks, that by reducing the number of models it increases the probability of classify and, therefore, recognize the correct model for a given sequence, the classifier must be loaded only with the needed models, instead of all of them. This fulfils the requirement R1. For that, in the initial phase, where the classifier is created, the cultural layer is responsible for returning a list of gestures to be recognized based on the commands that the application will accept. Therefore, this layer in the shamanic interface creates an abstract level for the system, in order to retrieve the full set of gestures to be recognized, based on the list of commands and the individual's culture, through a look up table between the pair command-culture to a gesture. The result of the mapping is a gesture name, that is later linked to the correspondent model. This creates a independence level in the cultural layer from the models used, fulfilling therefore the requirement R2.

The later result in the classifier it might be consider a similar approach to a simple gesture interface, where the recognized gesture is directly associated to a command. However, in this case a cultural notion was already recognized in a earlier part of the gesture recognition process. By creating a direct association between the gesture and the computational interaction, the time spent on every calculation of the command that is characteristic of a certain hand movement in a particular cultural knowledge described on the former architecture is almost non existent. On the other hand, this new approach creates a longer procedure in the initialization of a classifier, that only occurs in the start of a new state of the application or when the user changes the cultural background. Both of these cases always occur in introductory parts of the application, being already expected to take a bit longer while handling the user preferences. This will later create a more fluid interaction during the rest of the interface's functionality.

A simple map between a command in the system and a gesture name was also created, where a default gesture is associated a command, without none of both being connected to a specific culture, since some gestures does not look to be associated to a certain culture and being common between all cultures, or being only different in certain cultures, remaining equal between all of the others. This was noticed during the research for meaningful gestures, where it was found that each culture has a distinct and very restrict set of gestures and corresponding meanings. While it was also noticed during the analysis, that both users accomplished the same gesture for some commands.

## 4.6 Implementation in the game

The shamanic interface is implemented in the game through: a cultural layer that is responsible for storing all the following association between commands, gestures names and respective cultures; a map between all the gestures names and the gesture models (HMM in this case); and a variable to store the culture of the current player. After charging the cultural layer and linking all the gestures models to their respective names, in a presence of a defined culture and a well define set of commands used in the current state of the application, it is possible to initialize a classifier to recognize a given sequence of gestures.

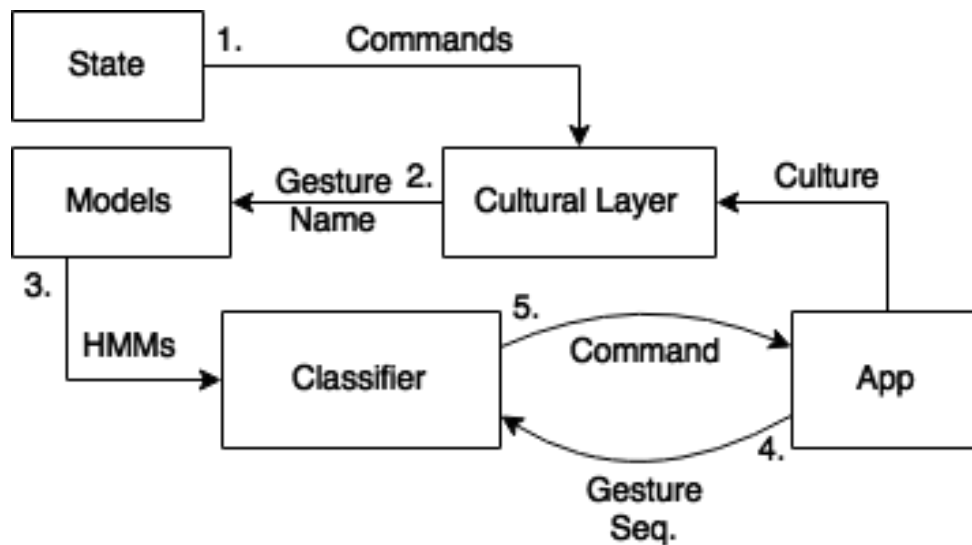


Figure 4.3: Control flow inside the system when creating and using a classifier

The application uses, as base for detecting and display the hand 3D models, the Unity scripts developed by *Leap Motion, Inc.*, where, after capture the current frame, each hand is separated, treated (created, updated or deleted in the absence of it in a new frame) individually and display it in the Unity scene. Preserving this approach, and since all the gestures required are performed with a single hand, a sequence buffer for each hand is created, where it stores each `Sign` associated to the hand in each consecutive frame, being deleted once the hand disappears from the read frame. Each hand is tracked individually by its ID, a unique parameter in the `Hand` structure which is the same during successive frames while the tracked hand remains visible. The adoption of a buffer per hand allows a user to perform two different commands at the same time, where each command is associated to one hand. By retrieving a list of buffer sequences present in all detected hands, it is possible to classify each sequence that prevails in each buffer and estimate which gesture is being made by each hand. Since the classifier starts with the gesture's models associated to the commands based on the individual's culture, in the beginning of a new state in the game, the returning set is a list of the user's desired interactions with the application.

Therefore the system implements abstraction of each distinct part in the game that requires a

## Approach

different set of commands to interact with it. By doing this, during the start of each state conditions, a classifier, capable of recognize all the gestures in the user's culture to control the game during the current state of it, is created. The game also recognizes the default gestures available in the *Leap Motion API*, more specifically the `ScreenTapGesture`, the `KeyTapGesture` and the `SwipeGesture`. These are used to select buttons in the menus or to play the game when a particular culture is not selected. While almost all the interactions in menus take advantage of the recognition of the `ScreenTapGesture` and the `KeyTapGesture` to press displayed buttons, independent of the selected culture, if any was chosen in the game level. If no culture was selected, the user can still play using these three default gestures. This creates an extra abstraction of interaction for the rest of the user that is not reached by the available cultures. The requirement R6 is, therefore, considered to be implemented.

During the gesture recognition, a small buffer of commands were created, in order to ensure that some gestures were performed during a certain period of time. This is achieved by storing the list of commands recognized in a buffer. Not only the command, but also the current time when it first appears are stored, and removed when the command does not appear in the given list of commands. Returning all commands that are in the buffer longer than the expect time. This allows the user to stop performing a gesture that is being miss recognized before it is processed by the application.

## Chapter 5

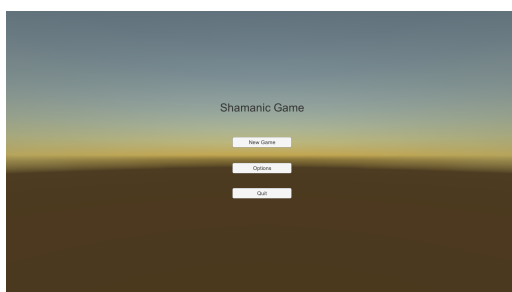
# Implementation

This chapter describes the implementation of the previous approach through several Design Cycles of a Design Science methodology, by explaining the different prototype created in each cycle of the application's development. In the end of each cycle an experimental test was performed with the purpose of improve the development path with the results and start a new design cycle with insights on the work made so far.

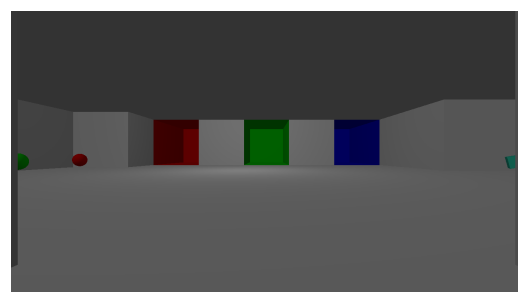
While in the following section is described an overall explanation of how the implementation occurred between prototypes, the conceived experimental tests and the results from them, the lasts sections contains the detailed information about the implementation in each class through several prototypes created.

### Prototype 0

In the very initial phase, it was developed a Prototype 0, where it was only implemented the game's structure and mechanics, in order to create a solid base application before any implementation of a shamanic interface. This prototype result showed that the application fulfilled the game's description. However, some visual feedback and graphical aspects needed to be improved, in order to create a more immersive sensation on the player. Figures in 5.1 show examples of screenshots during the experimental tests of this prototype.



(a) Main Menu



(b) Game Level

Figure 5.1: Screenshots of Prototype 0

## Implementation

### Prototype 1

During the creation of a first approach of a SI and its implementation in the current application, the main base of a library used in the application to create a shamanic interface was developed, as well as the program responsible to read and record frames, to be later used to teach the corresponding gestures' models. Hence, an initial shamanic interface was created in the game, using the libraries, and a small amount of gestures, namely moving forward/backward and rotating left/right, were recorded and the models created. An example of a gesture being recorded in the end of this prototype is in Figure 5.2, where, to support the visualization of the gesture being recorded, the *Leap Motion Diagnostic Visualizer* was used. After this, another set of experimental tests were executed. These targeted mainly the implementation of the Leap Motion SDK's plugins and assets, the correct association between cultural gestures and commands in the system, as well as the recognition of the few gestures recorded and modelled.

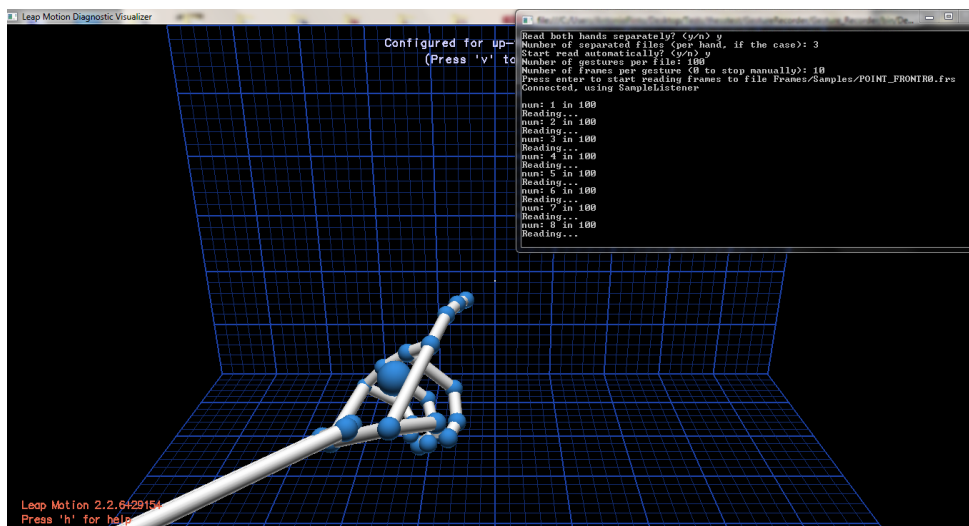


Figure 5.2: Recording the gesture POINT FRONT

As result of the tests on Prototype 1, it was possible to confirm Leap Motion's functionality to detect and track the user's hand to execute and record gestures. The application by recognizing the few commands implemented (i.e moving forward/backwards and rotating to the left/right) in the different cultures proved the correct functionality of the classifier, the cultural layer, when processing a map between commands and cultures to gestures, the created models and the record of gestures.

In the end, it was noted that, to recognize a wider range of gestures, the features stored and used to analyse and create models should be increased. It was therefore proposed to extend the hand's featuring by adding the palm's characteristics. With the change of models, and because the recording is a time consuming process, it was obvious that recording of gestures should be independent from the hand's featuring. Instead, by saving frames, it is not necessary to perform and record all gestures whenever a feature is changed in the method.



### **Prototype 2**

Based on the results from the previous prototype, a prototype 2 was created. Here, all commands were recorded, now in frames, modelled using a new feature selection and implemented in the game. After applying all the changes, the game level and the selection of some options (*e.g.* number of colours in the game) can be completed using gestures, but to select buttons in menus it was still needed the use of a mouse click. Therefore, the application is tested in order to evaluate the correct functionality and detection of all gestures in the game. By successfully passing the experimental assessment, the application could now enter in an end phase development, where some final modification took place to tune and prepare the system for a final experimental test and eventually an usability test with external users.

### **Prototype 2.5**

In a intermediate version, the addition a new value was implemented and tested to the hand's characteristics from which the models are created and analysed. This new selected feature was the variation of hand's position between frames. This modification was later refused, since the end result increased the chance of a gesture's miss recognition, mostly when static gestures were performed with certain motions.

### **Prototype 3**

In the last prototype, the default gestures available in the Leap Motion API were applied, allowing all users to click on buttons through gestures, as well as enabling a user with a non-available culture to play the game with a general set of gesture. Since the gestures used are only performed with one hand, the way the gestures were recognized also changed in this last prototype. Instead of recognizing the full frame as a simple gesture, each hand was separated from the frame and analysed individually, in order to recognize and produce a final command for each hand. The graphical aspect and the feedback given to the player about the environment around him in the level were also improved during the developing of this prototype.

The last batch of experimental tests was designed to assess all functionalities in the application, with a special focus on the new aspects of the game, such as the use of gestures in menus by clicking in buttons, and the use of non-cultural gestures to play the game. As result, the application recognized all gestures, by being capable to successfully complete, with all the three set of gestures, the game level, as well as navigate through the menus and select all the options available with a tap gesture instead of using the traditional mouse click.

The developed of these prototypes, a more detailed implementation where all the work is described, can be found in the appendix [C](#).

## Implementation

## Chapter 6

# Evaluation

In this short chapter is described the playability tests performed with users of different cultures. The experimental tests that were performed in the end of each design cycle can be found in the previous chapter.

After the experimental evaluation, a batch of playability tests, with the purpose of observing reactions and gathering feedback from users with different cultures. This test was executed in 3 groups of 4 peoples each, not only to test each cultural set, but also to analyse the interaction of user with the non-cultural set of gestures. Therefore the first group aimed users with a Portuguese culture and the second one users with a Dutch culture. Both where used to evaluate their own set of cultural gestures, while a third group composed by a miscellaneous of cultures, namely Dutch, Belgian, Indian and Brazilian, aimed at experimenting the third approach of interacting with the application.



Figure 6.1: Example a user performing a gesture

An example of a evaluation performed with a Dutch user (Figure 6.1) is available in <https://www.youtube.com/watch?v=O1xncT8HUz4> (camera) and <https://www.youtube.com/watch?v=FtFylxbUmrY> (screen).

## Evaluation

All users were able to complete the game level, but was noticed that, in the beginning, almost all users had some difficult in relating the hand position with the displayed hand model. After this initial impact with the gesture interface, the users were able to selected the respective culture (or non-culture) and the difficult of the game by selecting the number of colours presented in the map.

During the game level, it was noticed that, while some users tried to perform static gestures with a exaggerated motion, mainly in the movement gestures, others executed dynamic gestures, such as drink and grab gesture, without performing the complete motion and thus obtaining the result with a static sign that was similar to the dynamic one. It was also noticed that some gestures were sometimes miss recognized, this problem was especially visible in the gesture to move forward on the Dutch culture (Figure 6.2). With the default gestures provided by the Leap Motion API, it was noticed some difficulties performing them, since the parameters selected to achieve the gesture were equal for all users and each person had a different sensibility to perform the gestures.

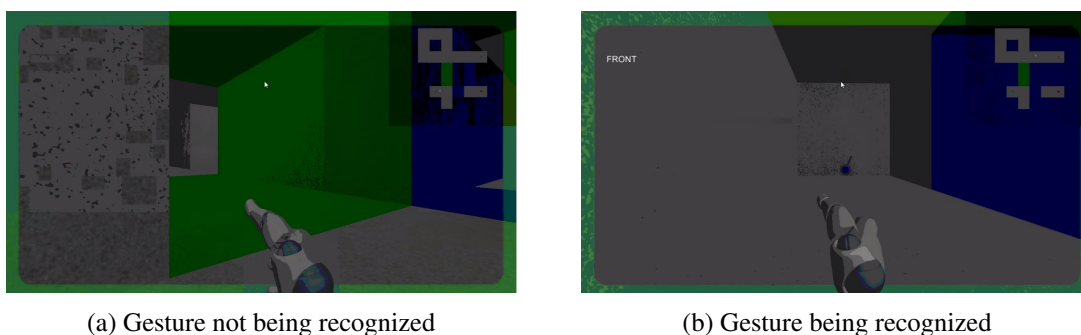


Figure 6.2: Moving Forward gesture

and that sometimes the displayed hand model did not correctly match the gesture performed by the user.

In the end of the evaluation, almost all users had said that they had experienced some difficulties at first, and that sometime the displayed hand model did not correctly matched the gesture performed by the user. Some users gave different suggestions for one or another gesture, when using the cultural set, stating that the rest of the gestures looked natural, while the users with the non-cultural set of gestures, said that they got accustomed to the patterns selected but felt a more mechanical aspect in the gestures rather an natural one.

A second playability test, aimed at having a small approach of how the memory of the user is affected when using a gesture recognition interface with meaningful gestures, was planned. However, this test was not possible to be realized due to time constrains. In a future, this is certainly an interesting evaluation to be performed.

# Chapter 7

## Conclusions

This last chapter consolidates all the work and present a future work section, where further paths for the developed work are discussed .

### 7.1 Results (Fulfilment of the Goals)

During the short duration of this thesis, a application was created based on a shamanic interface developed to recognize cultural gestures through the use of Hidden Markov Models and a classifier. With this approach, it was possible to improve the previous work on the shamanic interface architecture, by pre calculating the gestures needed and linking to the corresponding interaction in the system.

The developed project not only fulfilled the proposed requirements, but also the goals of this thesis. It has produced an application based on a shamanic interface, where users are able to use their own cultural set of gestures to interact with the application, and it expands the interface implementation based on the approach used to recognize different gestures.

### 7.2 Future Work

Being a shamanic interface a promising but recent aspect in the gesture recognition area, further work is needed. Not only to improve the research tool hereby created, but also by using it to further explore the concept of using it in order to explore the concept of using meaningful cultural gestures in a natural user interface.

#### **Test the shamanic interface concept**

Several tests might be performed by using the research tool developed, since one of the goals of this thesis was to develop a research tool that empirical researchers might use for this reason, testing. An example of a possible test is to find relations between the use of cultural gestures and the user memory on a long term of the gestures performed to achieve a specific command.

### **Generalize the interface**

As initially proposed, the interface should, not only, be independent from the application, but also from the capture the device and the modes used. Therefore, some implemented classes developed to support the SI, should be generalized by creating upper classes capable of being implemented later on any well defined system, device or model.

### **Improve the gesture recognition**

Although this thesis only touched the problem of gesture recognition using an approach based on Hidden Markov Models classification, several other methods and models for classification can be used. The hand featuring selection can be also subject for further project.

### **Apply Virtual Reality to the game**

One aspect in NUI's is the immersive experience that it offers to the user. Therefore, this application can increase the sense of immersion in the player by implementing virtual reality technologies such *Oculus Rift* [VR15] or *Google Glass* [Goo15] displays.

### **Use the SI concept in different paths**

Trying to identify the cultural background of a user through the use of a shamanic interface is another possible path of exploration, such as many others in this recent and wide concept of using meaningful cultural gestures in a natural user interface.

All the work developed during this thesis is available in the public repository

<https://github.com/Wolfox/SIGame.git>

# References

- [AMSP14] Muhammad R Abid, Philippe E Meszaros, Ricardo Fd Silva, and Emil M Petriu. Dynamic hand gesture recognition for human-robot and inter-robot communication. In *Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2014 IEEE International Conference on*, pages 12–17. IEEE, 2014.
- [BS12] Sigal Berman and Helman Stern. Sensors for gesture recognition systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(3):277–290, 2012.
- [dC14] Filipe Miguel Alves Bandeira Pinto de Carvalho. Shamanic interface for computers and gaming platforms. master thesis, FEUP, 2014.
- [dS12] César Roberto de Souza. A tutorial on principal component analysis with the accord.net framework. *arXiv preprint arXiv:1210.7463*, 2012.
- [dS14] César de Souza. Sequence classifiers in c# - part i: Hidden markov models. <http://www.codeproject.com/Articles/541428/Sequence-Classifiers-in-Csharp-Part-I-Hidden-Marko>, December 2014.
- [Goo15] Google. Google glass. <http://www.google.com/glass/>, June 2015.
- [Hev07] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.
- [HM03] Alan R Hevner and Salvatore T March. The information systems research cycle. *Computer*, 36(11):111–113, 2003.
- [HPKJ12] Guan-Feng He, Jin-Woong Park, Sun-Kyung Kang, and Sung-Tae Jung. Development of gesture recognition-based serious games. In *Biomedical and Health Informatics (BHI), 2012 IEEE-EMBS International Conference on*, pages 922–925. IEEE, 2012.
- [IK12] Noor Adnan Ibraheem and RafiqulZaman Khan. Survey on various gesture recognition technologies and techniques. *International journal of computer applications*, 50(7):38–44, 2012.
- [KSG<sup>+</sup>13] Michelle Karg, A-A Samadani, Rob Gorbet, Kolja Kuhnlenz, Jesse Hoey, and Dana Kulic. Body movements for affective expression: a survey of automatic recognition and generation. *Affective Computing, IEEE Transactions on*, 4(4):341–359, 2013.

## REFERENCES

- [MA07] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311–324, 2007.
- [MA09] MA Moni and ABM Shawkat Ali. Hmm based hand gesture recognition: A review on techniques and approaches. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 433–437. IEEE, 2009.
- [MAD14] M Mohandes, S Aliyu, and M Deriche. Arabic sign language recognition using the leap motion controller. In *Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on*, pages 960–965. IEEE, 2014.
- [McN92] David McNeill. *Hand and mind: What gestures reveal about thought*. University of Chicago Press, 1992.
- [MMC80] Peter Marsh, Desmond Morris, and Peter Collett. *Gestures, their origins and distribution*. Madison Books, 1980.
- [Mor14] Leonel Morgado. Cultural awareness and personal customization of gestural commands using a shamanic interface. *Procedia Computer Science*, 27:449–459, 2014.
- [Mot15a] Leap Motion. Api overview - leap motion c# sdk v2.3 documentation. [https://developer.leapmotion.com/documentation/csharp/devguide/Leap\\_Overview.html](https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html), June 2015.
- [Mot15b] Leap Motion. Leap motion | mac & pc motion controller for games, design, & more. <https://www.leapmotion.com/>, June 2015.
- [NPWZ14] Michał Nowicki, Olgierd Pilarczyk, Jakub Wasikowski, and Katarzyna Zjawin. Gesture recognition library for leap motion controller. Bachelor thesis, Poznan University of Technology, 2014.
- [Sua10] Daniel Suarez. *Freedom (TM)*, volume 2. Penguin, 2010.
- [SWTO04] Caifeng Shan, Yucheng Wei, Tieniu Tan, and Frédéric Ojardias. Real time hand tracking by combining particle filtering and mean shift. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 669–674. IEEE, 2004.
- [TFST14] João Marcelo Teixeira, Ronaldo Ferreira, Matheus Santos, and Veronica Teichrieb. Teleoperation using google glass and ar, drone for structural inspection. In *2014 XVI Symposium on Virtual and Augmented Reality (SVR)*, pages 28–36. IEEE, 2014.
- [VR15] Oculus VR. Introducing the samsung gear vr innovator edition. <https://www.oculus.com/>, June 2015.
- [WBRF13] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013.
- [WXK<sup>+</sup>09] Wang Wenhui, Chen Xiang, Wang Kongqiao, Zhang Xu, and Yang Jihai. Dynamic gesture recognition based on multiple sensors fusion technology. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 7014–7017. IEEE, 2009.



# Appendix A

## Game

### A.1 Description

In the game, the user must control a character through a maze, with the objective of finding its exit (grabbing an exit cube). Inside the maze some tiles of it may be colored (red, green or blue). The user to walk on it must *drink* first a potion of the same colour. The player can, therefore, only walk on the blocks with a neutral or with the last potion colour.

### A.2 Menus and Options

#### A.2.1 Main Menu

- New Game
- Settings
- Quit

#### A.2.2 Settings

- Number of colours (1,2 or 3)
- User Culture
- Back

#### A.2.3 Game

The user can:

- Move forwards
- Move Backwards

## Game

- Rotate Left
- Rotate Right
- Drink
- Grab
- Mute
- Unmute
- Pause
- Resume

### **A.3 List of Actions**

- Select
- Number 1
- Number 2
- Number 3
- Select Yes
- Select No
- Move Forward
- Move Backward
- Rotate Right
- Rotate Left
- Drink
- Grab
- Mute
- Unmute
- Pause
- Resume
- Quit

## Appendix B

# Cultural Gesture Analysis

It was performed a query targeting two persons with different cultures, Dutch and Portuguese, where it was asked to represent through a gesture the following words or actions:

- Select
- Number 1
- Number 2
- Number 3
- Yes
- No
- Move Forward
- Move Backward
- Rotate Right
- Rotate Left
- Drink
- Grab
- Mute
- Unmute
- Pause
- Resume
- Quit

## Cultural Gesture Analysis

The gestures obtained in response are presenting in the table [B.1](#), while the description of each gesture is the following:

- **Pointing** - pointing with the index
- **Index** - stretching only the index finger
- **Index+Middle** - stretching the index and middle finger
- **Index+Middle+Ring** - stretching the index, middle and ring finger
- **Thumbs Up** - thumbs up
- **Wave No** - waving with around 30° towards the floor (e.g "No, thanks")
- **Thumbs Down** - thumbs down
- **Open Hand Front** - pointing forward with the open hand
- **Pointing Front** - pointing with the index to the front
- **Pointing Back** - pointing with the thumbs to himself/herself
- **Open Hand Right** - pointing with the open hand to the right side
- **Pointing Right** - pointing with the thumb to the right side
- **Open Hand Left** - pointing with the open hand to the right side
- **Pointing Left** - pointing with the thumb to the left side
- **Mimic Glass** - moving the hand towards the user's lips whilst it is shaped like it is cupping a glass
- **Mimic Bottle** - thumb and pinky finger stretched while the hand moves against user's the lips
- **Grab** - closing the hand in middle air
- **Index Hush** - stretching the index in front of the lips
- **Mimic Mouth** - keeping all the fingers stretched with the thumb parallel and under the rest of the fingers. Move the thumb against all the other finger.
- **Halt** - all finger stretched with the palm of the hand pointing to the front
- **Rotate Hand** - all fingers stretched rotating around a horizontal axis with the hand's palm facing the user
- **Rotate Index** - index finger stretched rotating around a horizontal axis.
- **Wave** - all fingers stretched waving against the user.

## Cultural Gesture Analysis

Table B.1: Response gestures

<b>Action</b>	<b>Dutch person</b>	<b>Portuguese person</b>
Select	Pointing	Pointing
Number 1	Index	Index
Number 2	Index+Middle	Index+Middle
Number 3	Index+Middle+Ring	Index+Middle+Ring
Yes	Thumbs Up	Thumbs Up
No	Wave No	Thumbs Down
Move Forward	Open Hand Front	Pointing Front
Move Backward	Pointing Back	Pointing Back
Rotate Right	Open Hand Right	Pointing Right
Rotate Left	Open Hand Left	Pointing Left
Drink	Mimic Glass	Mimic Bottle
Grab	Grab	Grab
Mute	Index Hush	Index Hush
Unmute	Mouth Mimic	Mouth Mimic
Pause	Halt	Halt
Resume	Rotate Hand	Rotate Index
Quit	Wave	Wave

## Cultural Gesture Analysis

# Appendix C

## Detailed Implementation

### C.1 ShamanicInterface.Culture

#### C.1.1 CulturalPair

Since this C# library was developed in a .NET Framework 3.5 version, and the `Tuple` Class that provides a data structure with specific number and types of elements is only available from the version 4 in the .NET Framework, the `CulturalPair` Class was created to support the needed of a pair structure in the `CulturalLayer` Class.

Therefore, this class is presented with two single strings, a culture and a action, and the overridden methods `Equal()` and `GetHashCode()`, in order to be able to use it as a `Key` in the `Dictionary<TKey, TValue>` Class.

#### C.1.2 CulturalLayer

This class contains the cultural lookup tables where the shamanic interface stands out from others interfaces.

##### C.1.2.1 Prototype 1

In the first version of this Class, a `Dictionary<TKey, TValue>` is implemented , with a `CulturalPair` Class as a `Key` attribute and a string as a `Value` attribute(C.1). The methods responsible for adding or editing a `Key` and its respectively `Value` and other methods to obtain a `Value` based on a culture and a single command name, or a list of them.

```
1 private Dictionary<CulturalPair, string> culturalGestures;
```

Listing C.1: Cultural Gesture map

```
1 public List<string> GetGesturesNames(List<string> actions, string culture) {
```

## Detailed Implementation

```
2     return actions.ConvertAll(action => GetGestureName(action, culture));
3 }
```

Listing C.2: Obtain a List of Gestures

### C.1.2.2 Prototype 2

In the second version of this class, a default map is implemented on a `Dictionary<TKey, TValue>`, where both Key and Value are represented by a string, in order to store a gesture that does not have a cultural association, being common to the cultures (C.3). A new method to add or edit a Key and its Value in this new map is added, as well as the previous method of adding and retrieving a gesture with a cultural meaning is altered. If a cultural gesture is not found in the `culturalGestures` variable, the retrieving method (C.4) looks for it in the `defaultGestures`. In the adding method (C.5) of a cultural gesture, if the custom gesture is not defined for the command, both dictionaries increment the same combination, the custom gesture of this command may be altered at a later time..

```
1 private Dictionary<string, string> defaultGestures;
```

Listing C.3: Default Gesture map

```
1 public void AddCultureGesture(string action, string culture, string
   gestureName) {
2     if (!defaultGestures.ContainsKey(action)) {
3         AddDefaultGesture(action, gestureName);
4     }
5
6     CulturalPair cultPair = new CulturalPair(action, culture);
7     try {
8         culturalGestures.Add(cultPair, gestureName);
9     }
10    catch (ArgumentException) {
11        culturalGestures[cultPair] = gestureName;
12    }
13 }
```

Listing C.4: Adding a Cultural Gesture

```
1 public string GetGestureName(string action, string culture) {
2     CulturalPair pair = new CulturalPair(action, culture);
3     string value;
4     if (culturalGestures.TryGetValue(pair, out value)) {
5         return value;
6     }
```



## Detailed Implementation

```
6     }  
7     return defaultGestures[action];  
8 }
```

Listing C.5: Obtain a Gesture

## C.2 ShamanicInterface.DataStructure

### C.2.1 Sign

This class represent the single set of the desired values in a gesture.

#### C.2.1.1 Prototype 1

In this version, the class is simply composed by an array of doubles (C.6).

```
1     private double[] values;  
2  
3     public Sign(int dim = 0) {  
4         values = new double[dim];  
5     }
```

Listing C.6: Values variable and the constructor in the Sign Class

#### C.2.1.2 Prototype 2

In the second version, the class adds a method to retrieve the length of the array (C.10).

```
1     public int GetDimensions() {  
2         return values.Length;  
3     }
```

Listing C.7: Get dimensions method in the Sign Class

### C.2.2 Sequence

This class is represent the full gesture of a user through the time, having therefore a set of Sign.

#### C.2.2.1 Prototype 1

As the previous class, this class is composed by a simple set of Signs (C.8).

```
1     public List<Sign> sequence;
```

---

### Listing C.8: Sequence of Signs

#### C.2.2.2 Prototype 2

In this version, a method is added to confirm the continuous size and retrieve the dimensions of the `Sign` elements in the sequence (C.10).

```
1 public int GetDimensions() {
2     if (sequence.Count < 1) { return 0; }
3     return sequence[0].GetDimensions();
4 }
5
6 public bool CheckDimensions() {
7     int dim = GetDimensions();
8
9     for (int i = 1; i < sequence.Count; i++) {
10        if (sequence[i].GetDimensions() != dim) { return false; }
11    }
12
13    return true;
14 }
```

Listing C.9: Get and check dimensions methods in the `Sequence` Class

#### C.2.3 SequenceList

This class is initially created to support the recording, and later the learning part.

##### C.2.3.1 Prototype 1

It contains a list of `Sequence`. In a first approach, this class is created to help the recorder. Therefore a `Save` and `Load` methods are created (C.11).

##### C.2.3.2 Prototype 2

While adding a method to retrieve the dimensions in the `Sign` elements of the diverse `Sequence` on this class(C.10), both methods of saving and loading are removed (by commenting them) since they are not longer necessary.

```
1 public int GetDimensions() {
2     if (sequences.Count < 1) { return 0; }
3     return sequences[0].GetDimensions();
4 }
```

Listing C.10: Get dimensions method in the `SequenceList` Class

```

1  /*public void Save(Stream stream) {
2      IFormatter formatter = new BinaryFormatter();
3      formatter.Serialize(stream, this);
4  }
5
6  public static SequenceList Load(Stream stream) {
7      IFormatter formatter = new BinaryFormatter();
8      SequenceList obj = (SequenceList)formatter.Deserialize(stream);
9      return obj;
10 }*/

```

Listing C.11: Save and Load method in the `SequenceList` Class

## C.2.4 SequenceBuffer

This class is used to store the latest  $X$  elements of a continuous reading from the capture device. Therefore it contains a `Sequence` class and a maximum size. When adding a new `Sign` in the buffer, if the current size exceeds the maximum size, it removes the first element from it (C.12).

```

1  public void AddSign(Sign sign) {
2      if (sign != null) {
3          buffer.sequence.Add(sign);
4      }
5
6      while (buffer.sequence.Count > bufferSize) {
7          buffer.sequence.RemoveAt(0);
8      }
9  }

```

Listing C.12: Add method in the `SequenceBuffer` Class

## C.3 ShamanicInterfaceClassifier

### C.3.1 HMMClassifier

This class represent a classifier of Hidden Markov Models, also with the respective models and the names associated to each model. In order to use this class, after adding all models in the class, the classifier must be initialized through the `StartClassifier` (C.14) method before computing the most likely model given a sequence (C.13).

## Detailed Implementation

```
1     public int ComputeToInt(double[][] sequence) {
2         return classifier.Compute(sequence);
3     }
4
5     public string ComputeToString(double[][] sequence) {
6         return names[ComputeToInt(sequence)];
7     }
```

Listing C.13: Methods used for compute a sequence

```
1     public void StartClassifier() {
2         classifier = new HiddenMarkovClassifier<
3             MultivariateNormalDistribution>(models.ToArray());
4     }
```

Listing C.14: Initialization of the classifier

## C.4 ShamanicInterface.State

### C.4.1 Actions

This class contains a set of string, in order to store the commands used in each part of the application.

## C.5 ShamanicInterface.Utils

This static class contains a miscellaneous of static methods, responsible for helping the interaction between the library and external applications. The methods presented in this class can be divided into the following subjects.

### C.5.1 Transform Leap Motion Data into SI Data

A group of methods was conceived to create instances of the `Sign` class from the `Frame` and `Hand` classes in the Leap library for C# available in the Leap Motion SDK, as well for all the consecutive classes, such as `List<Frame>` to `Sequence` and `List<List<Frame>>` to `SequenceList`.

#### C.5.1.1 Prototype 1

Initially the conversion between these two kind of data, is executed from a single `Frame`. This results in a single `Sign` composed by the direction of the five finger available from the farthest to the right hand of the frame.

### C.5.1.2 Prototype 2

In a second version, in the previous function refer is added the direction of the hand to the values inserted in the `Sign` class.

### C.5.1.3 Prototype 2.5

In an intermediate prototype, the variation of the position is added. This feature is removed after the number of miss classifications of models given a sequence increase.

### C.5.1.4 Prototype 3

In this final version, the extraction of hands characteristic is migrated to a single method, therefore being able to create a `Sign` from the `Hand` class.

## C.5.2 Save and Load Frames and HMMs

This part in the class `Utils` is responsible for saving and loading both the frames stored with the recorder and the HMM created from it. While in the *Prototype 1* the methods to save and load the HMM are only presented (C.15), since these were created from instances of the `SequenceList` class in the first prototype. During the developed of *Prototype 2* the methods to save and load frames are created (C.16), in order to be used during the creation of gesture models.

```

1      public static void SaveHMM(HiddenMarkovModel<
2          MultivariateNormalDistribution> model, string path) {
3          model.Save(path);
4      }
5
6      public static HiddenMarkovModel<MultivariateNormalDistribution> LoadHMM
7          (string path) {
8          return HiddenMarkovModel<MultivariateNormalDistribution>.Load(path)
9          ;
10     }

```

Listing C.15: Methods to save and load a HMM

```

1      public static void SaveFrame(Frame f, string path) {
2          byte[] serializedFrame = f.Serialize();
3          System.IO.File.WriteAllBytes(path, serializedFrame);
4      }
5
6      public static Frame LoadFrame(string path) {
7          byte[] frameData = System.IO.File.ReadAllBytes(path);
8          Controller control = new Controller();
9          Frame f = new Frame();

```

## Detailed Implementation

```
10         f.Deserialize(frameData);
11         control.Dispose();
12         return f;
13     }
```

Listing C.16: Methods to save and load a single Frame

### C.5.3 Creation of gesture models

This part contains a single method, `CreateModelFromFrames` (C.17), that creates, teaches and returns a Hidden Markov Model with the a list of frames that contain all the given sequences of the same gesture to be used in the learning algorithm applied here.

```
1     public static HiddenMarkovModel<MultivariateNormalDistribution>
2         CreateModelFromFrames(List<List<Frame>> frames) {
3         SequenceList sequences = Utils.FramesToSequenceList(frames);
4
5         HiddenMarkovModel<MultivariateNormalDistribution> hmm;
6         MultivariateNormalDistribution mnd = new MultivariateNormalDistribution(
7             sequences.GetDimensions());
8         hmm = new HiddenMarkovModel<MultivariateNormalDistribution>(new Forward(5),
9             mnd);
10
11         var teacher = new BaumWelchLearning<MultivariateNormalDistribution>(hmm);
12         teacher.Tolerance = 0.0001;
13         teacher.Iterations = 0;
14         teacher.FittingOptions = new NormalOptions()
15         {
16             Diagonal = true, // only diagonal covariance matrices
17             Regularization = 1e-5 // avoid non-positive definite errors
18         };
19         teacher.Run(sequences.GetArray());
20         return hmm;
21     }
```

Listing C.17: Method used to create a HMM

### C.5.4 Acquirement of gestures' models

Based on a cultural layer presented in this implementation (the culture of the user, a list of commands, and a list with all gesture models), the methods developed (C.18) create a list of gesture models that represent commands in the user's culture.

## Detailed Implementation

```
1     public static List<HiddenMarkovModel<MultivariateNormalDistribution>>
2         GetModelsWithCulture(
3             Dictionary<string, HiddenMarkovModel<MultivariateNormalDistribution>>
4                 allModels,
5             List<string> actions, CulturalLayer cultureLayer, string culture = "") {
6         return GetModels(cultureLayer.GetGesturesNames(actions, culture), allModels
7             );
8     }
9
10    private static List<HiddenMarkovModel<MultivariateNormalDistribution>>
11        GetModels(
12            List<string> gestureNames,
13            Dictionary<string, HiddenMarkovModel<MultivariateNormalDistribution>>
14                allModels) {
15        return gestureNames.ConvertAll(gestureName => allModels[gestureName]);
16    }
```

Listing C.18: Methods used to obtain a list of models of gestures

## C.6 Recorder

### C.6.1 Program

This is the main class of the Recorder, where all the methods that allows the user to define which gesture is going to record or aggregate are defined, as well as the parameters of how the recording will occur.

#### C.6.1.1 Prototype 1

In this initial prototype, the user could record and aggregate gestures with only a specific group of parameters, being able to change it by editing the code of the project.

#### C.6.1.2 Prototype 2

In order to remove the need to constantly alter the code all the times that it is expected to record or aggregate a gesture with different reading parameters, an initial basic interface through command lines was implemented. In this version, the user is already capable of selecting the settings in which the reading will occur.

#### C.6.1.3 Prototype 3

In the final version, the aggregate method join sequences of both hands alternately, instead of sequentially.

## C.6.2 Recorder

This class is in charge of managing the Listener, as well as store the end result of each reading process in an individual file.

### C.6.2.1 Prototype 1

It contains the methods used to create and manage a `RecordListener` based on the user preferences to record the gesture. The methods to store recorded frames in a single file were also developed.

### C.6.2.2 Prototype 2

In the second version, the recorder start saving set of `Frames` instead of instances of the `SequenceList`.

```

1  while (state != RecorderState.Saving) {
2      Console.WriteLine("num: " + (actualNumOfReads+1) + " in " + numOfReads);
3      if (!isAuto) {
4          Console.ReadLine();
5      }
6      state = RecorderState.Reading;
7      Console.WriteLine("Reading...");
8
9      while (state == RecorderState.Reading) {
10         if (numOfFramesPerRead == 0) {
11             Console.ReadLine();
12             listener.GetSequence();
13         }
14     }
15     while (state == RecorderState.Storing) { }
16 }
17 while (state != RecorderState.Saving) { }
18
19 Utils.SaveListListFrame(sequencesToRead, path);

```

Listing C.19: Cycle to manage multiple records of the same gesture in a single file

## C.6.3 RecordListener

A Listener class, inherited from the *Leap Motion* library, it overrides the `OnFrame` method, that occurs when a new frame is captured, in order to store it.

### C.6.3.1 Prototype 1

It contains the methods used to capture and store the `Sign` of a captured `Frame`.



### C.6.3.2 Prototype 2

In the final version, the `Frame` is not transformed into a `Sign` any more. Only the latter is captured, since it contains all the physical characteristics of the hands.

```

1  public override void OnFrame(Controller controller)
2  {
3      if (parent.state != Recorder.RecorderState.Reading) { return; }
4
5      Frame frame = controller.Frame();
6      HandList hands = frame.Hands;
7      Hand hand = hands.Rightmost;
8      if (!hand.IsValid) { return; }
9      //if (hands.Count > 1) { Console.WriteLine("MORE THAN 1 HAND"); return; }
10
11     sequence.Add(frame);
12     if (sequence.Count >= numOfFramesPerSeq && numOfFramesPerSeq != 0)
13     {
14         GetSequence();
15     }
16 }
17
18 public void GetSequence()
19 {
20     parent.state = Recorder.RecorderState.Storing;
21     parent.Store(sequence);
22     sequence = new List<Frame>();
23 }

```

Listing C.20: Overridden method `OnFrame` in `RecordListener` Class

## C.7 Application - Maze Game

### C.7.0.3 Prototype 0

In this prototype the mechanics behind the game were all developed, such as the menu flow, the creation of the game level and all the aspect in it, and the controls and interactions of the character with other objects in the game. In this stage, the game was able to be controlled through a keyboard and a mouse.

### C.7.0.4 Prototype 1

This prototype already contains the implementation of a initial SI, where it was only possible to control the player's movements with cultural gestures, the rest of the game, including the menus, was controlled with the same input devices from the previous prototype.

### C.7.0.5 Prototype 2

In this phase of the game, where the change of the components is represented in the model, the models can now be recreated by using recorded frames instead of the previous Recorder list of sequences. A HUD is also introduced in the game in order to provide feedback to the user about the interaction with the game environment.

### C.7.0.6 Prototype 3

The final prototype changes the way that the frame is interpreted into commands, by implementing the `handGesture`, a class where it has a `SequenceBuffer` to store the sequences of gestures performed by each hand. With this, the prototype can recognize gestures per hand instead of only recognize a single gesture in the all frame. The default gestures available in the `Leap Motion SDK` are also implemented, so players with a culture that is not available in the interface can play the game without being forced to use a set of gestures created from a foreign culture. With this implementation, it is also possible to use the default gestures `KeyTapGesture` and `ScreenTapGesture` to press in the menu buttons.