Universidad Carlos III de Madrid

Systems engineering and automation department

BACHELOR THESIS

Gesture Recognition with Kinect and machine learning techniques

Author: Arturo Arranz Mateo

Tutor: Tomás de la Rosa Turbides

Co-Tutor: Raquel Fuentetaja Pizán

Madrid, October 2015

Título: Gesture Recognition with Kinect and machine learning techniques
Autor: Arturo Arranz Mateo
Tutor: Tomás de la Rosa Turbides

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día .... de .... de ... en ...., en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Echando la mirada atras parece que fué ayer cuando empece esta aventura en el mundo de la ingeniería. Termino esta etapa en la Carlos III con no solo un educación sino con infinidad de recuerdos, experiencias y amigos que me acompañaran el resto de mi vida.

Agradezo, como no puede ser de otra forma, a mis padre, Pedro y Remedios, que sin su apoyo y cariño nada de esto habría sido posible. Siempre han procurado y esforzado en que tanto mi hermano y yo tengamos un buen futuro. Esfuerzo que sin duda es impagable.

Agradezco a mi hermano por ser sin duda un gran pilar de mi vida. Persona a la que acudo siempre que necesito consejo, y siempre lo recibo. Le agradezco sin duda el entusiasmo que desde siempre me ha contagiado pro la ciencia.

También quiero agradecer a mis amigos de la carrera que han hecho de este, un divertido camino. En especial quiero agradecer a Irene, Jesús, Celia y Juan con los que tan estrechos lazos he creado.

Por último quiero agradecer a mi tutor y cotutora, Tomás y Raquel, por no haber dudado en aceptarme como tutores cuando me presente a las puertas de su deprtamento. Por supuesto agrdezco también su ayuda y paciencia conmigo.

Resumen

Este proyecto describe el diseño e implementacion de una plataforma de reconocimento de gestos usando un sensor de movimento y técnicas de aprendizaje automático. La solcuión pretende probar diferentes algoritmos para reconocimento de gestos, así como crear una plataforma flexible y reutilizable.

Para desarrollar el proyecto las últimas tendencias en Interacción Humano Máquina y detección de movimiento han sido analizadas a fondo.

Palabras clave: Interacción Humano-Máquina, HCI, Aprendizaje Automático, Kinect, Kinect para Windows, SDK, SVM, GNB, KNN, LDA, Python, Scikit-Learn, C#.

Abstract

This Project describes the design and implementation of a gesture recognition platform using a motion detection device and machine learning techniques. The project is focused on testing different algorithms on gesture recognition, as well as creating a flexible platform which could reusable.

In order to develop the project the latest trends on Human-Computer interaction and motion detection have been analyzed thoroughly.

Keywords: Human Computer Interaction, HCI, Machine Learning, Kinect, Kinect for windows, SDK, SVM, GNB, KNN, LDA, Python, Scikit-Learn, C#.

Contents

List of Figures

Chapter 1

Introduction

Human - computer interaction(HCI) is directly linked with the evolution of computers and the necessity of controlling and communicating with them, so we can conclude that born with the first computers.

We have to go back to 1943, to the first computer, the ENIAC (Electronic numerical integrator and computer). It was developed by the research department of the USA balistic army. The ENIAC computer filled 167 square meters and the input/output interaction was done through punch cards (primitive form of HCI) as we can see in the Figure 1.1. Fortunately, HCI, as well as computers, evolved.

Was not until 1948, with the IBM SSEC(Selective Sequence Electronic Calculator) that computers could receive instructions from a electro-mechanical registers, and not until 1951 with the UNIVAC 1 that computers could give information in form of blinking lights.

Between 1960 and 1970 the teletype monitor dominated HCI, which was a monitor where the output of the computer was displayed.

Before 1970 only technology professionals and dedicated hobbyists where the only humans who interacted with computers. But this changed with the raise of personal computers. The amount of people interacting with machines experimented a huge growth during this time, and also the HCI. Two main invents played a huge role. The development of the first cheap

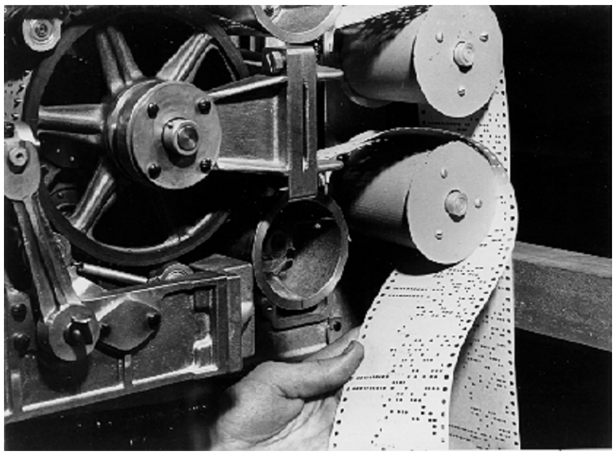


Figure 1.1: Input/Output mechanism from the ENIAC

Figure 1.2: Qwerty keyboard, used nowadays

composite monitors by Don Lancaster, Lee Felsenstein and Steve Wozniak in 1976 and QWERTY keyboard, as the on in the Figure 1.2 in late 80s that is sill used nowadays.

Another boom from the last years is the tactile screen which is dominating the market of mobiles, touch pads, and recently personal computers.

However, the evolution is still on going and nowadays there is several fields of research and exploration. 3D recognition is an example, which is already widely used in veideogames, but is also making his way in other areas, for instance the project Soli from Google.

The advances of input and output devices created academic interest on the communication between human and computer. Joseph Carl Robnett Licklider[8], one of the fathers of HCI said in 1960:

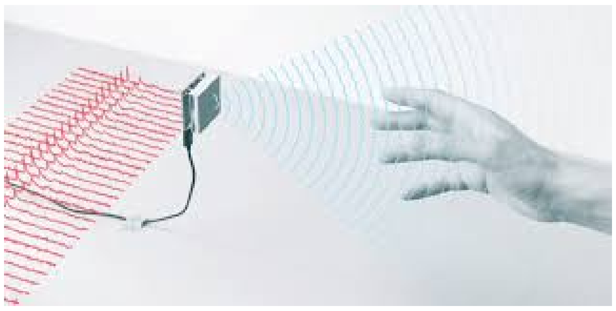


Figure 1.3: Prototype of Soli project

" The hope is that, in not too many years, human brains and computing machines will be coupled together very tightly, and that the resulting partnership will think as no human brain has ever thought and process data in a way not approached by the information-handling machines we know today."

This affirmation is not far from the truth. Everyday the technology is more embedded in our daily life. HCI is driven by the necessity of getting this technology closer to humans needs. Poor interfaces are rejected by general public, who prefer simplicity over complexity. Gesture recognition, the topic of this thesis, is about simplifying the interaction, using natural language used in the daily basis.

1.1 Objectives

We have seen the importance of Human computer interaction and how this should be simplify to have more natural interaction machines. We want to explore the possibility of controlling machines through gestures used in the daily basis.

We would be make this gesture recognition method flexible. A platform where new gestures can be added easily, so the users or developers can use for their own purposes. For doing so, we will use machine learning algorithms, which allow to introduce this new gestures through a training phase.

The objective of this bachelor thesis, as the title state, is **recognize gestures with machine learning techniques.** This would consist in tow different phases: (1) a way to extract the physical data from the users and (2) a technique to recognize the gestures.

For the first part we will need to create a platform that allow us to store and record the gestures to generate our own dataset.

For the second part, as we said, we will need to apply and test different machine learning algorithms to find which one adapt better to our problem.

1.2 Document Structure

The following document is organized in different chapters and sections. Each one of these chapters will rivet on different subjects. A brief explanation on the chapters will be made below.

- The first chapter (this one) will give a brief **Introduction** aimed to explain the background and history on Human Computer Interaction as well as set the goals for the project.
- The second chapter will be the **State of the Art** of the project. Here will be explained the current used techniques for gesture recognition as well as a brief intro to the sensor and machine learning library used.
- In the third chapter the **Proposed Solution** will be explained as well as the development.
- In the fourth chapter the proposed solution is **Evaluated** and the results are presented.
- The fifth chapter, **conclusions** are presented reviewing the goals set in the beginning. Also some ideas on how the project can be amplify are presented, for **Future works**.
- The sixth chapter, **Project Management and Budget**, the initial planning is compared with actual process and times, and the reasons

for the differences are discussed. Also the total prize for the project is presented.

- At the end of the document we can find the **References** used for the project.

Chapter 2

State of the art

2.1 Introduction

Human gestures recognition consist in identifying and interpreting automatically human gestures. Gesture Recognition can be generally divided in two sub-problems: (1)the gesture representation problem and (2) the pattern recognition problem. In this section state of the art about gesture representation, gesture recognition, will be presented, as well as the technologies involved.

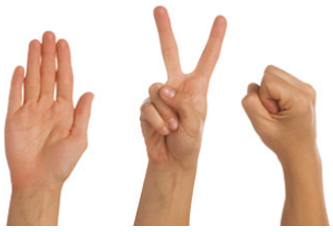
2.1.1 Definition and nature of gestures

A gesture can be defined as a body movement. A gesture is a non-verbal communication, used instead or in combination with a verbal communication. It can be articulated with any part of the body such as hands, head, eyes, etc. Gestures are a very relevant part of the humans communication and interaction. However, same gestures can have very different meaning in each culture. For example, moving the head side to side, commonly means ”no” in occidental cultures, while in south Asia, especially India, means ”OK”, ”good” or ”understood”.

Gestures can be categorized in many ways. If gestures are used to express information, according to Rime and Schiaratura[2] they can be dis-

(a) Symbolic gestures

(b) Pantomimic gestures


(c) Iconic gestures

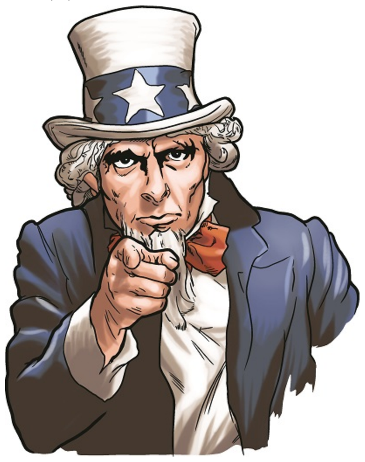

(d) Deictic gesture

Figure 2.1: Gestures used to express information

tinguished as:

- Symbolic: Gestures that have a single meaning in the given culture. For example the thumbs up gesture.
- Deictic gestures: pointing gestures which can be concrete (pointing to a real location, object or person) or abstract (pointing abstract location, period of time).
- Iconic: it consists of hand movements that depict a figural representation or an action (hand moving upward with wiggling fingers to depict tree climbing).
- Pantomimic: These are gestures that consist in using imaginary objects or tools. Mimicking the use of a music instrument.

Gestures can also be categorized as dynamic or static. In the latter case, the gesture becomes a posture. In addition, we can categorize gestures by the part of the body imply on the movement: hand gestures, hand/face gestures, body gestures. In the present project we aim to recognize **dynamic body gestures**.

2.1.2 Technologies

In this subsection we overview the enabling technology for gesture recognition.

- Wired Globes: can give to the computer the orientation and position of the hands using inertial and magnetic sensors. Wired globes give very accurate tracking, but lacks in friendly use.
- Controller-based: These devices are extensions of the body, so when the gesture is performed its tracked by the device. A clear example is Wiimote, the controller of the Wii console. Another example is the mouse of the computer.
- Single Camera: A standard camera can be used for gesture recognition. However this lack the depth aware, only 2D space can be detected. However, single cameras are still used because the cheap price.
- Stereo Cameras: Using two or more cameras, and knowing the distance between them, depth can be inferred.
- Depth-aware cameras: Using specialized cameras such as structured light or time-of-flight cameras, one can generate a depth map of what is being seen through the camera at a short range, and use this data to approximate a 3D representation of what is being seen.
- Radar: Google new project Soli, aim to capture gestures with a tiny radar device. The project is still under development.

2.2 Gesture Representation

There are several models to represent the human body, but they can, generally, been split in two categories:(1) 3D models and (2) appearance based models.

2.2.1 3D based models

The 3D based model define the spatial 3D position of the body. Generally, there are three kinds of representation models.

- Volumetric model: these approaches have been heavily used in computer animation industry and for computer vision purposes. The models are generally created of complicated 3D surfaces.
- 3D geometric models: These model are less precise than volumetric in terms of skin surface information, but still have essential skeleton information.
- 3D Skeleton model: the body is abstracted in a skeletal representation by tracking relevant joints. This high abstracted representation reduce the computational weigh, while keeping the basic structural information of the body.

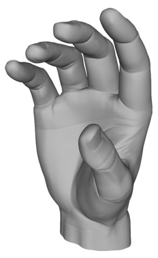


Figure 2.2: Volumetric hand model

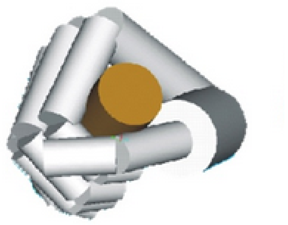


Figure 2.3: 3D Gemoetric hand model

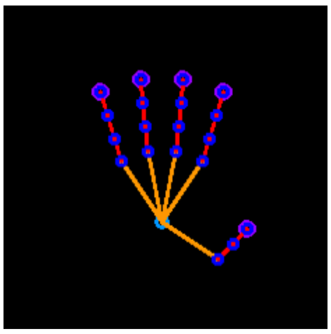


Figure 2.4: 3D skeleton hand model

2.2.2 Appearance-based methods

These models don't use a spatial representation of the body anymore, because they derive the parameters directly from the images or videos using a template database. Some are based on the deformable 2D templates of the human parts of the body, particularly hands. Deformable templates are sets of points on the outline of an object, used as interpolation nodes for the object's outline approximation. One of the simplest interpolation function is linear, which performs an average shape from point sets, point variability parameters and external deformators. These template-based models are mostly used for hand-tracking, but could also be of use for simple gesture classification.

2.3 Approaches for Human Gesture Recognition

Once physical data is extracted successfully it is necessary to recognize gestures. We will explain the two main approaches for this issue: Automata-based approaches and machine learning techniques.

2.3.1 Machine learning algorithms

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system[13]. These are:

- **Supervised:**The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- **Unsupervised:**No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end.

- **Reinforcement:** A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal or not. Another example is learning to play a game by playing against an opponent.

Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system[3]:

- **Classification:** inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one (or multi-label classification) or more of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".
- **Regression:** also a supervised problem, the outputs are continuous rather than discrete.
- **Clustering:** a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- **Density estimation:** finds the distribution of inputs in some space.
- **Dimensionality reduction:** simplifies inputs by mapping them into a lower-dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.

In our case we attempt to recognize a set of gestures, so in this case our problem can be treated as a classification problem. The typical procedure for this methods are the following: (1)feature extractions (i.e specific body points), (2) learning phase, where input training data is used train the model, and (3) the classification phase where the model is used to make predictions in unseen data. The classifiers algorithms used in this project are K-Nearest Neighbours (K-NN), Support Vector Machines (SVM), Gaussian Naive Bayes (GNB) and Linear Discriminant Analysis(LDA).

Gaussian Naive Bayes

The naive Bayes classifiers are based on the Bayes Theorem with a naive assumption that every pair of feature is independent with each other. Given a class variable y and a dependent vector x with n features the Bayes Theorem states:

$$Posterior = \frac{prior \times likelihood}{evidence}$$

This algorithm also assume that the features distribution is Gaussian.

Linear Discriminant Analysis

LDA is a non-parametric linear technique for classification. This method works by modelling the class conditional distribution P(f1,....fn— C) into a normal distribution, as the GNB does. Also each class Gaussian are assume to share the same covariance matrix. This lead to a linear decision boundary when comparing log-probability rations log(P(C=u— X)/P(C=k—X).

These classifier are an attractive choice since have a closed-form solution which can be easily computed.

K Nearest Neighbours

K-nn is a lazy non-parametric algorithms. In contrast to eager, lazy algorithms do not make generalization with training data, in other words there is not explicit training phase or it is very minimal. While other algorithms make generalizations and ride off of part of the training data, K-nn algorithm use all the training data to make predictions. However the the testing phase is very costly in term of time and memory.

The K-nn classifier labels an input sample Y according to the following procedure: firstly it finds the K nearest vectors to Y among the training data. This is usually performed with euclidean distances. Secondly it label Y as the majority votes class of these nearest neighbours.

Support Vector Machines

Support Vector Machines is a binary classifier that aim to find the hyperplane over the feature space that maximize the margin, which is the distance from closer points to the hyperplane. This lead to a optimization(maximize margin) problem given some constraints(i.e. classify correctly), which is solved by Lagrange multipliers.

There is several hyperparameters available for tuning the classifiers and avoiding overfitting. One of them is C, which trade off misclassification for smoother decision boundary, by allowing outliers not being classify. Multiplying a point by C, could be moved in the feature space in order to make it linearly separable. If C is too small any value could be moved and the decision boundary will not adjust to the real dataset. If C is too big, the classifier would be too regularized and will not find a optimal solution.

Another very powerful resource of SVMs is the *kernel transformations*. If the input space is not linearly separable, it can be transformed in a higher dimension space through the kernel function. Two common used are the polynomic expansion and the radial basis function(rbf). Both add hyperparamters to the system, *degree* for the polynomic kernel and *gamma* for rbf.

2.3.2 Automata-based approaches

Besides learning models, automata-based models are a very used methods for gesture recognition. For instance FSMs (Finite State Machines), HMM (Hidden Markov Model), DTW (Dynamic Time Warping) are kind of automaton with a set of states and a set of transitions. The states represent static gestures(postures) and the transitions represent next allowed changes with temporal and probabilistic constraints. Then a dynamic gesture is considered as a path between a initial state and a final state. The main disadvantage of automata-based algorithms is that once that we want to recognize a new gesture the model have to be change, making them not very flexible. Furthermore, the computational complexity is huge since normally it is proportional to the number of gestures to be recognize.

2.4 Sensor used: Microsoft Kinect

Microsoft Kinect is a movement control device that was original created for playing with the XBOX 360 console without the necessity of any type of remote controller. This way the player use his own body to interact with the game, creating a more realistic user experience.

Even thought that the sensor have been developed by Microsoft, the design and technology was created by the Israeli company PrimeSense. Kinect was announced for first time on 1 of June of 2009 at the Electronic Entertainment Expo 2009 with the name ”Natal project”. It was finally released on November of 2010 as a accessory of the XBOX 360 console.

Kinect was a great opportunity for researchers, hobbyists and inventors to make new applications and use the potential of this sensor for a very low cost in comparison with other technologies. In 2010 PrimeSense launched the first no official SDK, OpenNI.

The first Microsoft reaction was fight the hackers, but soon they realized that freeing the source code would generate enormous quantity of tools for the users. So, in 2011 Microsoft launched a beta version of the official SDK, and later in 2012 the SDK version 1.0.

Kinect is widely used nowadays in several areas, not only video games. A great example is the medicine applications such as the NAOtherapist[7] project developed in the Carlos III University, for rehabilitation tasks.

2.4.1 Hardware

The hardware contain four main parts that we can see in the Figure 2.5 and we explain below:

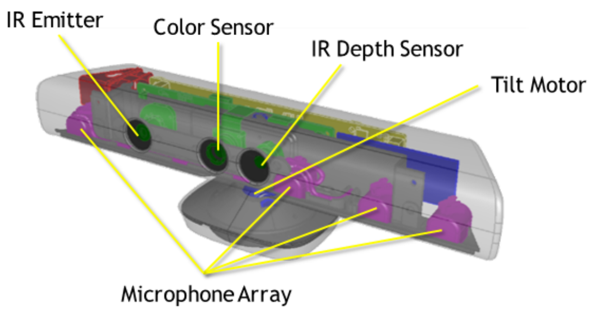


Figure 2.5: Microsoft Kinect sensor

- **3D depth sensor**. For the depth sensing an IR emitter and a IR sensor. The distance is calculated in function of the time that the light takes to reflect.
- **RGB camera**, which helps to identify images and videos with a maximum resolution of 1024x768 at 30fps.
- **Microphone array** used for speech recognition.
- **Tilt motor**, to move automatically up and down. It have a maximum inclination of ± 27degrees.

2.4.2 Basic Image recognition

The optical configuration of Kinect allow image recognition in real time. The technology used is not very complex. It is something that have been used since 20 years ago. However, Microsoft have developed a system that before was too expensive.

The sensor is compound by 2 main parts, as mentioned before, an infrared emitter and a infrared receiver. The depth of the objects is computed thanks to the beams reflections, creating a ”depth map” that allow the sensor differentiate between static objects and the people using it.

With the data from this "depth map", the software apply filters to know when a moving object is a person or not. This is inferred by simple rules like "a person have two legs".

Once that the information is organized, the body parts are identified and a skeleton is created. When a body part is not visible, it is inferred, thanks to the predefined positions that Kinect software have pre-stored. Al this is computed at 30fps.

2.4.3 Sensor specifications

In the following table we can see the technical specifications about the sensor to get a better insight and know how to get a better use.

Kinect sensor specifications	
Vision range	43∘ range vertical, 57∘ range horizontal
Tilt motor range	± 27∘
Frames speed	30fps
Depth resolution	640x480
RGB resolution	1024x768
Audio format	16KHz, 16bit PCM
Audio input features	Array of four microphones with 24 bits ADC and signal processing with echo cancellation and background noise suppression.

Table 2.1: Kinect Specifications

Another specification to consider is the distance range. It have two modes: *Default mode* and *Near mode.* We can see the optimal working distances for each mode in the Figure 2.6

2.4.4 The libraries

Here we will analyse the two main libraries for Kinect. In one side we have the official version SDK 1.8 and in the other side we have an open source version, OpenNI.

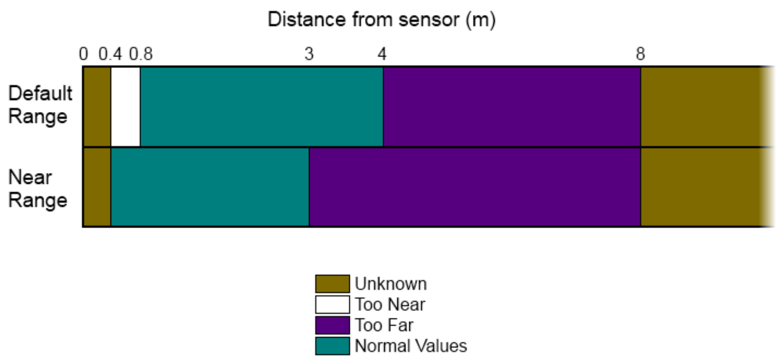


Figure 2.6: Kinect distance range

Microsoft SDK 1.8

Developed by Microsoft specifically for controlling Kinect. It can only be used with Windows operative system and the languages allowed are C++, C# and VB, so it is a little bit restrictive. The SDK includes the drivers for the computer, interfaces for the device with documentation from the developers and examples from the source code. We can track up to 6 people that are in the angle of the sensor, and use the 4 audio integrated sensors.

OpenNI

OpenNI is an API to develop applications which use Natural Interaction (Open Natural Interaction). The API cover the communication between low level sensors(Audio, vision..) and high level devices(like visual tracking).

OpenNI was written and distributed as GNU LGPL (Lesser General Public License) which means that the code is freely distributed and open for the geneeral public.

OpenNI is no official SDK, which is can be used with Kinect and much more devices with similar purposes. It was developed in 2009 and one of the main companies behind it was PrimeSense.

However,in November 2013, Apple bought PrimeSense for 350 million of dollars[1], and shutdown the original OpenNI project. Immediately after the shutdown, organizations that used OpenNI subsequently preserved documentation and binaries for future use. Today, Occipital and other former partners of PrimeSense is still keeping a forked version of OpenNI 2 (OpenNI version 2) active as an open source software for their Structure SDK for their Structure Product.

Comparison

Kinect for Developers and OpenNI comparison depicts the results of a comparison between the different features of the two libraries. The comparison has been carried out taking into account the previous consideration as well as the analysis perform by Hinchman[6]

	Microsoft SDK	**OpenNI**
Skeletal tracking	Yes	Yes
Joints	20	20
Calibration	No	Yes
Predictive Joint Tracking	Yes	No
Resolution	1024x768	800x600
Driver Installation	Easy	Hard
Languages supported	C++, C# and VB	Python, C, C++, C#, Lsip and more
Multiplatform	No	Yes
Documentation	Well documented and Official Forum	No official forum. Mailing list and twitter.

Each library have pros and cons. Even though that OpenNI give us more flexible in terms of platforms and language we have chosen **Microsoft SDK**. The main reason for this choice have been the good documentation and community from the official SDK. Also been extent from calibrating the sensor is a good point for making a easier to use product.

2.4.5 Architecture SDK v1.8

The official SDK provides a sophisticated software library and tools to help developers to use the Natural User Interface (NUI) API. In the Figure 3.1 we can observe teh basic interaction between the sensro, software and the applications.

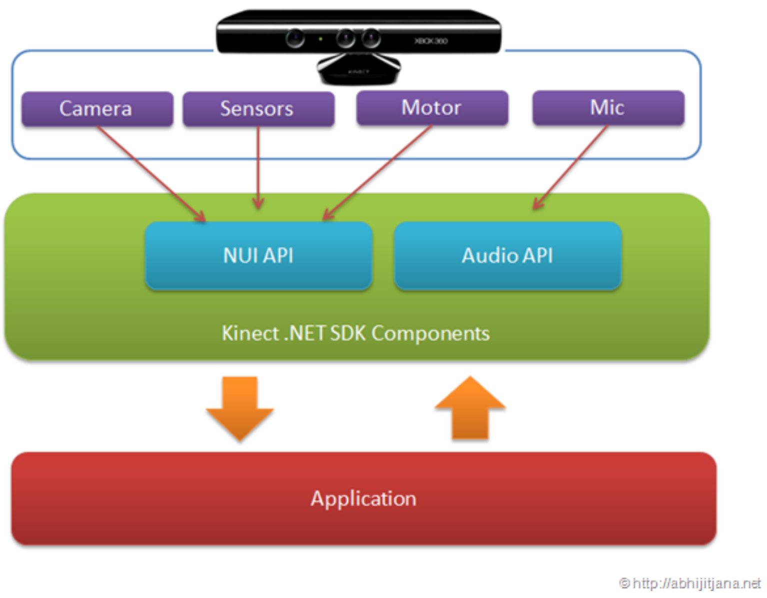


Figure 2.7: Sensor, software and applications interaction

From the basic scheme we can infer four main parts:

- **Hardware Kinect.** Include the components mentioned before (RGB camera, IR emitter...) and the USB cable which connects the sensor to the PC.
- **Kinect drivers.** The controllers of Windows for the Kinect sensor are installed within the the SDK installation. This controllers support:
  - The microphone system, as well as a kernel-mode which we can access through the Audio API.

  - Image and depth data streaming
  - Enumerate functions which allow to connect more than one Kinect to the same PC.

- **KinectAudio DirectX Media Object (DMO).** The KinectAudio DMO support the microphone system to localize the sound origins.

In the Figure 2.8 we can more specifically all the SDK architecture. It can be divided into in four big parts: hardware, kernel-mode, user mode and application.

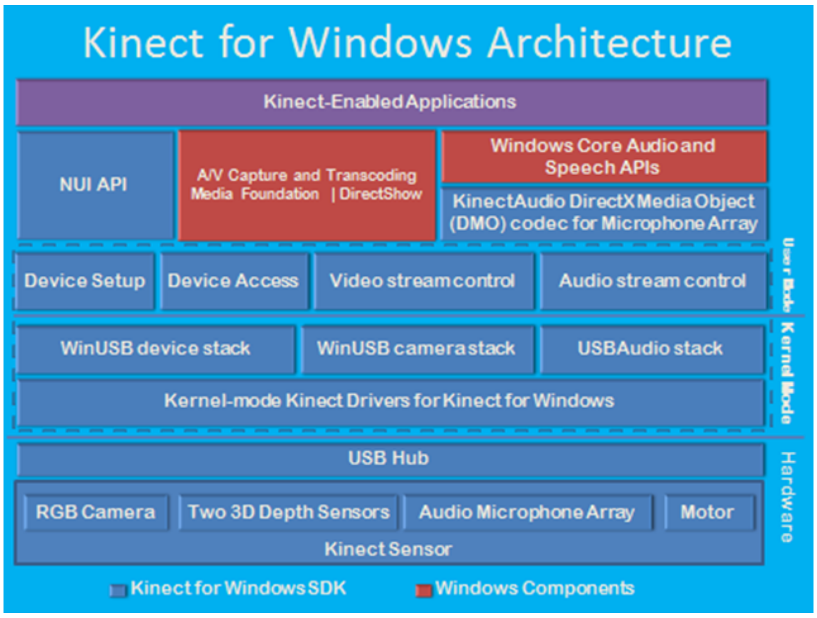


Figure 2.8: SDK architecture for Microsoft Windows

2.4.6 NUI API

The NUI API is the core of Kinect for Windows. This is in charge of the image processing and the device controlling. Include the following functions:

- Access to the different Kinect sensors which are connected to the PC.
- Access to the image and depth data streaming.

- Image and depth data processing to support the *skeletal tracking*

NUI initialization

As said, the Kinect controllers allow the use of multiple Kinects sensors in the same computer. Additionally, the NUI API include functions to enumerate the sensors, to determinate how many sensors are connected, take the IF of each one and configure the data streaming of each one individually. The NUI API have 4 subsystems, which could be configured in each application. Each of them could be selected at the initialization. This subsystems are:

- **Color**. Activate the RGB camera streaming data.
- **Depth**. Activate the depth sensors streaming data.
- **Depth and player Index.** Activate the detection of depth data and assign a ID to each player in the area.
- **Skeletal Tracking.** Generate and track skeletal structures of each player.

Image streaming data

The NUI API give us means to modify the Kinect configuration, giving us access to the streaming sensor data. The data streaming is a succession of static images. When initialize we can choose the type of data that we want to receive.

Color image data

The color image data is available in two different formats:

- **RGB**It provides bit maps of 32 bits with the format X8R8G8B8 (which reserve 8 bits for each color). To specify this format is necessary to use

- **YUV color.** It provides 16 bit maps with gamma correction. Using 16 bits per pixel, use less memory and reserve smaller buffer when in the data streaming. This format is only supported for 640x480 resolution at 15fps.

For choosing each format when opening the data streaming it is necessary to specify *color* or *color_YUV*. The Kinect sensor use a USB connexion to transfer the data to the PC, but this connexion have a limited bandwidth. The sensor return a 1280x1024 image, which is compressed and converted to RGB before the transmission. Once that that the transmission is done, the data is decompressed before transmitting to the application. The use of this compression allow us to work at 30fps, however the compression algorithm have a quality image loss.

Depth data

Each depth data streaming give us frames where each pixels represent the Cartesian distance between the sensor and the closer object in each x and y coordinate in the vision range. Each pixel in the streaming its represented with 13 bits for the data and another 3 bits to identify each player. A 0 value indicate that there is not depth data available in that position because the object is too close or too far. When the *skeletal tracking* option is not activated the 3 bits which identify the player have a 0 value.

Skeletal tracking

The NUI Skeleton API provide the position up to 6 players. In this information the orientation and pose of the player is also provided. This information is given as points, called *Joints*, which create a skeleton structure. It provide 20 *Joints*. In the Figure 2.9 we can check the tracked joints.

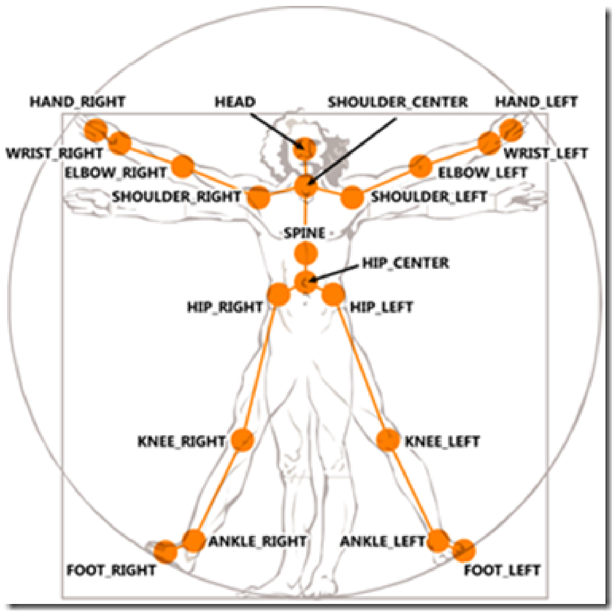


Figure 2.9: Detected joints by Skeletan tracking

NUI transformations

Each depth data frame have a resolution of 640x480, 320x240 or 80x60 pixels of size. Each one represent the Cartesian distance, in millimeters, from the plane of the camera to the closer object in the x and y position. If a pixel have 0 value means that there is no object in the kinect range. In the Figure 2.10 we can see the depth coordinate system.

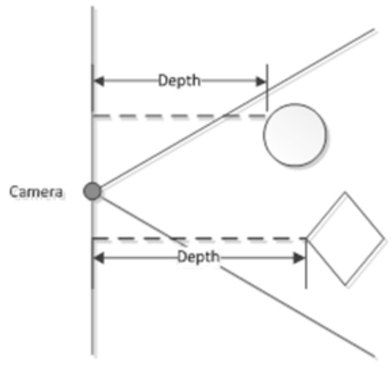


Figure 2.10: Depth data coordinate system

The skeleton position data of the users are expressed in a x,y and z coordinate system, in meters. The center of coordinates are the kinect sensor, with the positive z axis towards the direction where the sensor is facing. The positive y axis go up, while the positive x axis go to the left as shown in the Figure 2.11

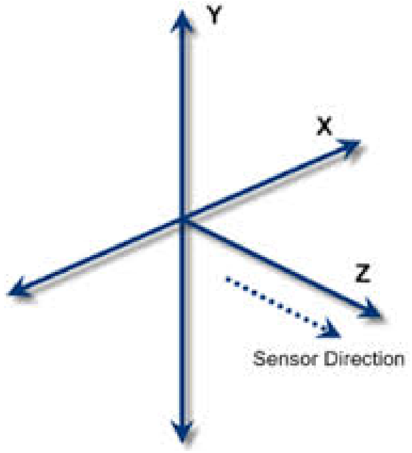


Figure 2.11: Coordinate system for skeleton data

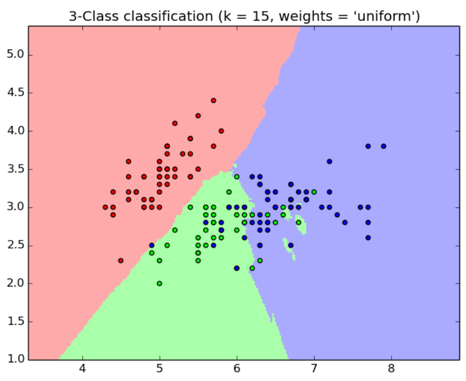
2.5 Machine Learning library: scikit-Learn

Scikit-learn is a open source machine learning library with a commercially usable license[11] implemented in **Python**.The project started in 2007 as a Google Summer of Code project by David Cournapeau. Matthieu Brucher started work on this project as part of his thesis. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel of INRIA took leadership of the project and made the first public release, February the 1st 2010. Since then, several releases have appeared following a $\tilde{3}$ month cycle, and a thriving international community has been leading the development.

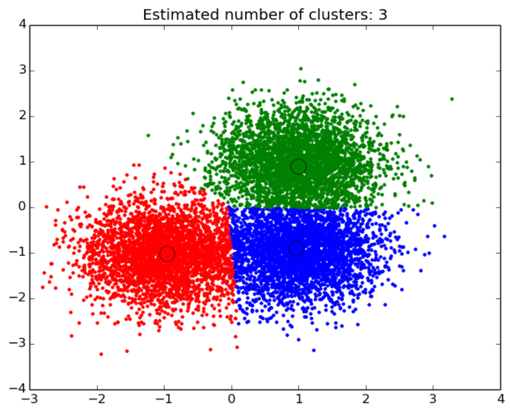
The project get funded by donations and the support of well known companies as Google, Inria and the Paris-Saclay Center for Data Science.

Scikit provides tools for most of machine learning techniques. This is some of them:

- **Classification:** Identify to which category an object belongs to. This would be our goal, classify weather it is a gesture or another. Scikit



(a) K-NN classification algorithm



(b) Mean-shift clustering algorithm

Figure 2.12: Machine learning algorithms in scikit-learn

have several classification algorithm implemented such as *SVM, K-NN, Decision Trees, Random Forest, Adaboost..*

- **Regression:** Predicting a continuous-valued attribute associated with an object. Some applications would be predicting stock prices, drug responses, etc.

- **Clustering:** Automatic grouping of similar objects into sets. Some algorithms implemented are *k-means, spectral clustering, mean-shift, etc.* An example of use would be customer segmentation.

- **Dimensionality reduction:** Reducing the number of random variables to consider. Algorithms implemented are, *PCA, feature selection, non-negative matrix factorization.* This techniques are used to improve the efficiency of the algorithms.

- **Model Selection:** Comparing, validating and choosing parameters and models.

- **Preprocessing:** Tools for normalizing and feature extraction. Transforming input data such as text for use with machine learning algorithms.

In the Figure 2.12 we can see some machine learning algorithm implemented in sckit-learn. Scikit is also widely used in the industry for prestigious companies such as Spotify, Evernote, and DataRobot. The great community behind the project, the well documentation and the ease for reusability have been what made us choose this library.

Chapter 3

Proposed Solution

In this chapter we will define the problem as well as the solution. We will analyze the technical problems and how we will approach them.

3.1 Definition of the problem

The problem to solve is **recognizing dynamic gestures in a 3D space**. The gestures chosen, would have the purpose of controlling a interactive screen from the distance. Those gestures will be the following:

- Swipe slide left
- Swipe slide right
- Zoom in
- Zoom out
- Point in order to select something

There are several problems to solve: (1) First we have to get 3D data, due the nature of the chosen gesture, so we need a way to **sense depth** as well as width and length. (2) Secondly, it will be necessary record data during certain time since the movements are dynamic. As each gesture will be a succession of postures on time, one of the biggest problems to face would be the **feature extractions**. (3) And finally we will need a **pattern recognition** algorithm for the features extracted.

Nonetheless, not all the parts of the body will be necessary to track, since all can be perform the hands and arms.

3.2 Proposed solution

As explained, we can differentiate between two stages for our problem. In one side the extraction of physical data and on the other side the feature extraction and pattern recognition.

- **Data extraction:** We propose to use Kinect with the official SDK, well explained in the-state-of-art section. We will also generate our own dataset, recording and generating labeled gestures.
- **Pattern recognition:** we will use machine learning with the sckit-learn library in Python. We will treat this as a classification problem. Hence, we will implement supervised machine learning algorithms.

We will follow the standard procedure for supervised machine learning[10]. (1) preprocessing, (2) attribute extraction, (3) train model with our generated dataset, (4) testing model. The main reason for choosing machine learning over atomata-based (DTW, HMM, FSM..) is the flexibility we have for learning new gestures. We will generate a platform where training the models with new gestures would be easier, giving the project a reusability factor. Also **several models would be tested** in order to compare which one match better the given data. The algorithms that would be used are the ones explained in the-state-of-art chapter: SVM, GNB, KNN and LDA.

In the Figure 3.1 we overview the proposed architecture and in the following sections each part will be explained more deeply.

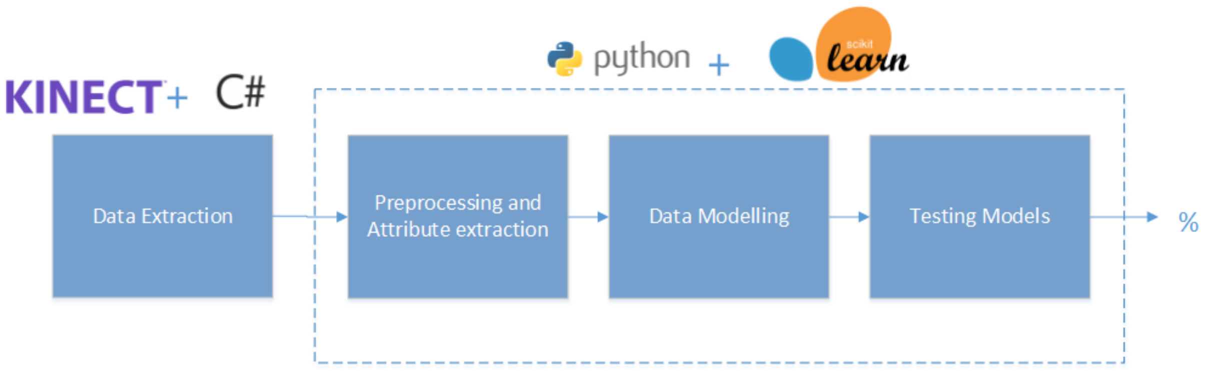


Figure 3.1: Architecture proposed

3.3 Data extraction

The data extraction would consist in two parts: (1) creating a platform for recording the gestures and (2) generating a dataset by recoding samples.

3.3.1 Recoding platform

For creating the platform, as said we will be using Kinect and the official SDK, which provides examples coded in C#, C++ and VB. We will *Skeleton Basics-WPF* coded C# and modify for our purposes. The main features that the example include are:

- Track up to 6 human bodies
- Track 20 joint from each body
- Interface for visualize skeletons in real time

The additional modification should provide a way to store the necessary joints data, and do during the interval where the gesture is performed. We want to highly the importance of the last part. If the gesture is not limited between the start and end we will be including not relevant information in the samples which will result in a poor performance of the machine learning algorithms. To reach our purposes the following changes have been made in the *Skeleton Basics-WPF* example :

- **Start/Stop button**: each time that a frame is ready it will check if it should be recorded or not. If so, it will be written in a CSV file skeletonData.csv. When the button is pressed and the program was in "recording" state it will generate a carriage return, which will indicate that a gesture have been performed.
- **Label selector**: combobox used to labeling the gesture that is being recorded. When recording, the label selected is written on the file with each frame.
- **Recording Signal**: a textbox which change the color for knowing when we are recording.

The Figure 3.2 is the interface with all this modifications implemented. Note that when recording not all the joints will be stored. As stated, only hands and arms are involved in the chosen gestures. However, the spine and head will be also stored to normalize the body size. Each frame will have the following data:

- LeftHand Joint: 3 coordinates
- RightHand Joint: 3 coordinates
- LeftElbow Joint: 3 coordinates
- RightElbow Joint: 3 coordinates
- Spine Joint: 3 coordinates
- Neck Joint: 3 coordinates
- Label: 1 string

As explained in the state-of-the-art, the depth data taken have 3 dimension, hence 3 coordinates per joint. The data is given in meters with the Kinect as the origin of coordinates.

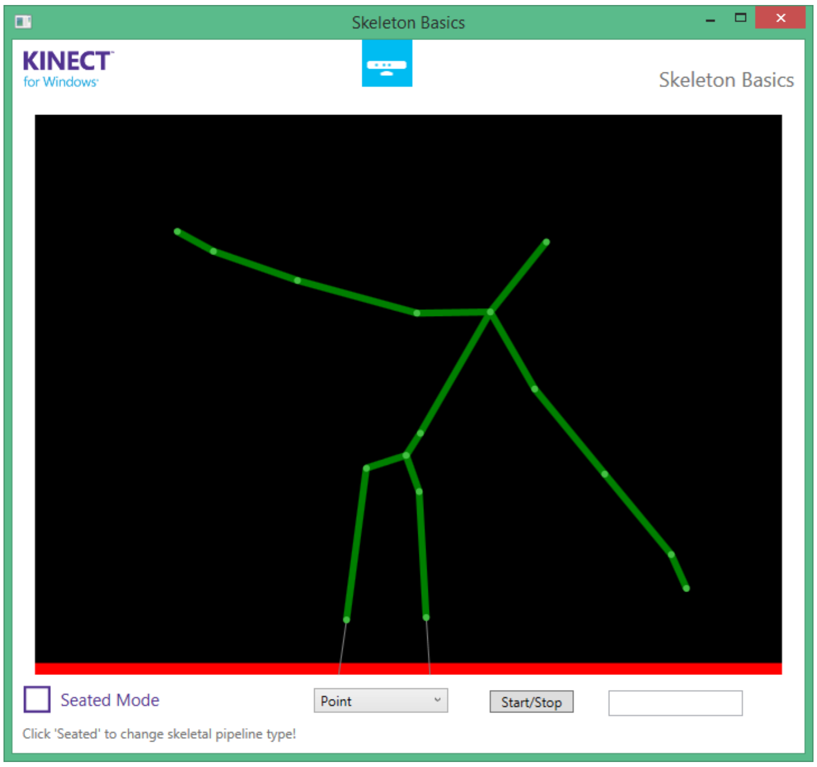


Figure 3.2: Skeleton basics modified interface

3.3.2 Generate dataset

The second part of the data extraction is, in fact, taking the samples. It is necessary a volume of samples for training and testing our algorithms. Better results will arise with bigger datasets. However taking too much samples would be impractical for generating new gestures. Another important part of sampling is the **diversity**. We have to generate the dataset with different people and performing the same gesture in different ways. This is necessary for generalizing, so the models are able to work for people with different sizes and with different manners of performing the same gesture. In the Figure 3.3 we can see a user taking samples of the selected. Finally, our dataset we consist in 504 samples that we can separate in:

- 108 samples of *Zoom Out*
- 101 samples of *Swipe Right*

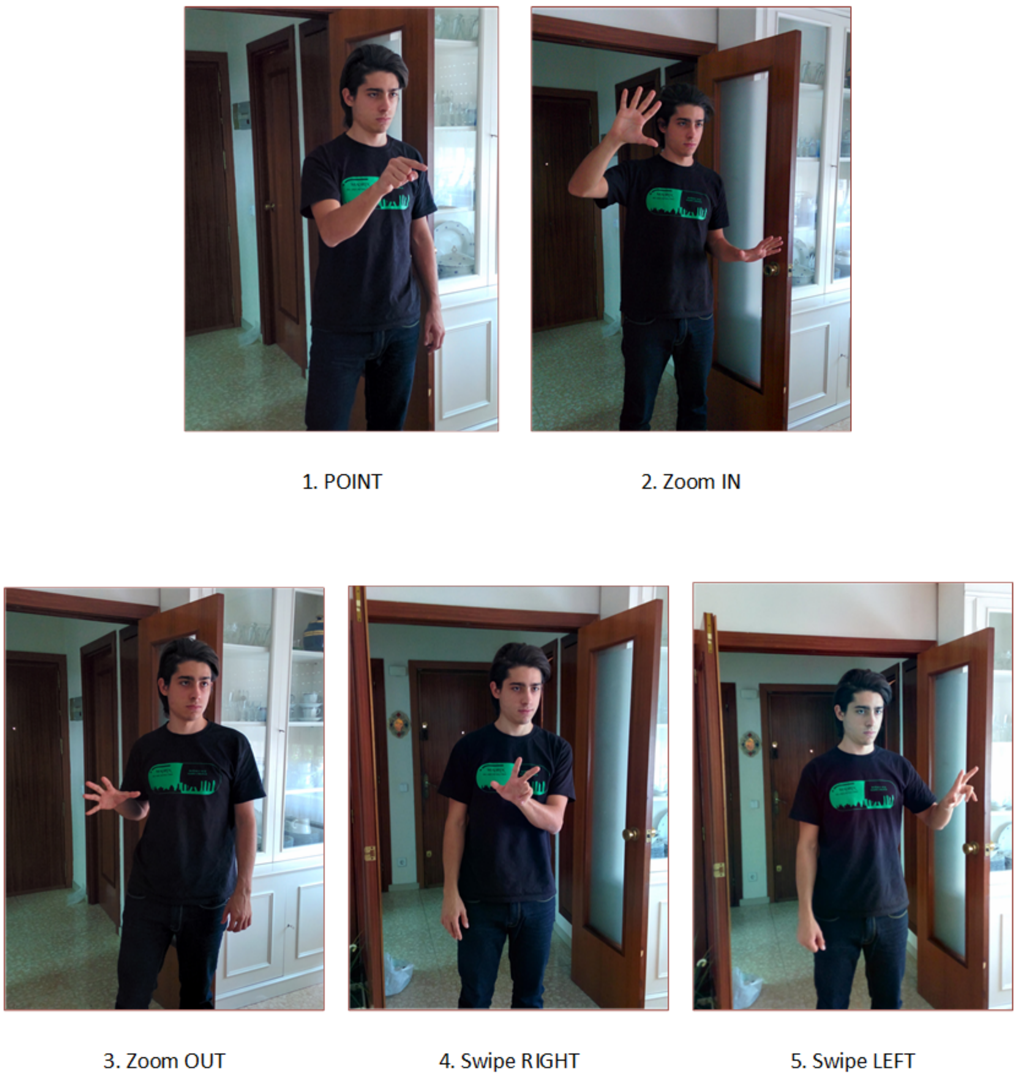


Figure 3.3: Performing each of the selected gestures

- 134 samples of *Swipe Left*
- 81 samples of *Zoom In*
- 79 samples of *Point*

3.4 Preprocessing Data

Once that physical data has been extracted we have to clean and prepare for the training methods. As explained in the Figure 3.4 this will consist in

(1) loading the extracted data, (2) normalizing, (3) extract attributes and (4) standardize. As said in the intro for machine learning (Preprocessing, Training and Testing), Python language have been used.

3.4.1 Load data

The data extraction phase has a CSV file as a output. In that file each gesture is defined as:

$Gesture = (N_{frames} \times 6_{joint} \times 3_{coordinates}) + 1label$

The frames per gesture is undefined since we do not know how long it take (around 80 frames). However, each frame is defined by the following values: 6 joints per 3 coordinates plus 1 label which sum 18 floats and 1 string. For processing the information a class, SkeletonFrame, have been created. Hence, a list of gestures, compound of a list of SkeletonFrames, is the output of the coded method *loadgestures(skeletonfile)*.

3.4.2 Normalize body size and center

To make indifferent from the body size every gesture is normalized. For doing so, the distance between the spine and the neck [12] have been used. In each gesture the mean neck-spine distance is calculated. Then every coordinate is normalized with a normalized factor, calculated from the neck-spine distance. Now, we can ride off of the spine joint and neck joint.

$Gesture = (N_{frames} \times 4_{joint} \times 3_{coordinates}) + 1label$

3.4.3 Attribute extraction

As we said at the beginning of this chapter, one of the biggest problem we had to face was extracting attributes for dynamic gestures. In other words, how to define a gesture on time. Our proposed solution is define each gesture as **differential movements**. Each succession of points have been chunked in n parts. Then is calculated the distance moved between the beginning and end of each part. This is done by calculating the distance

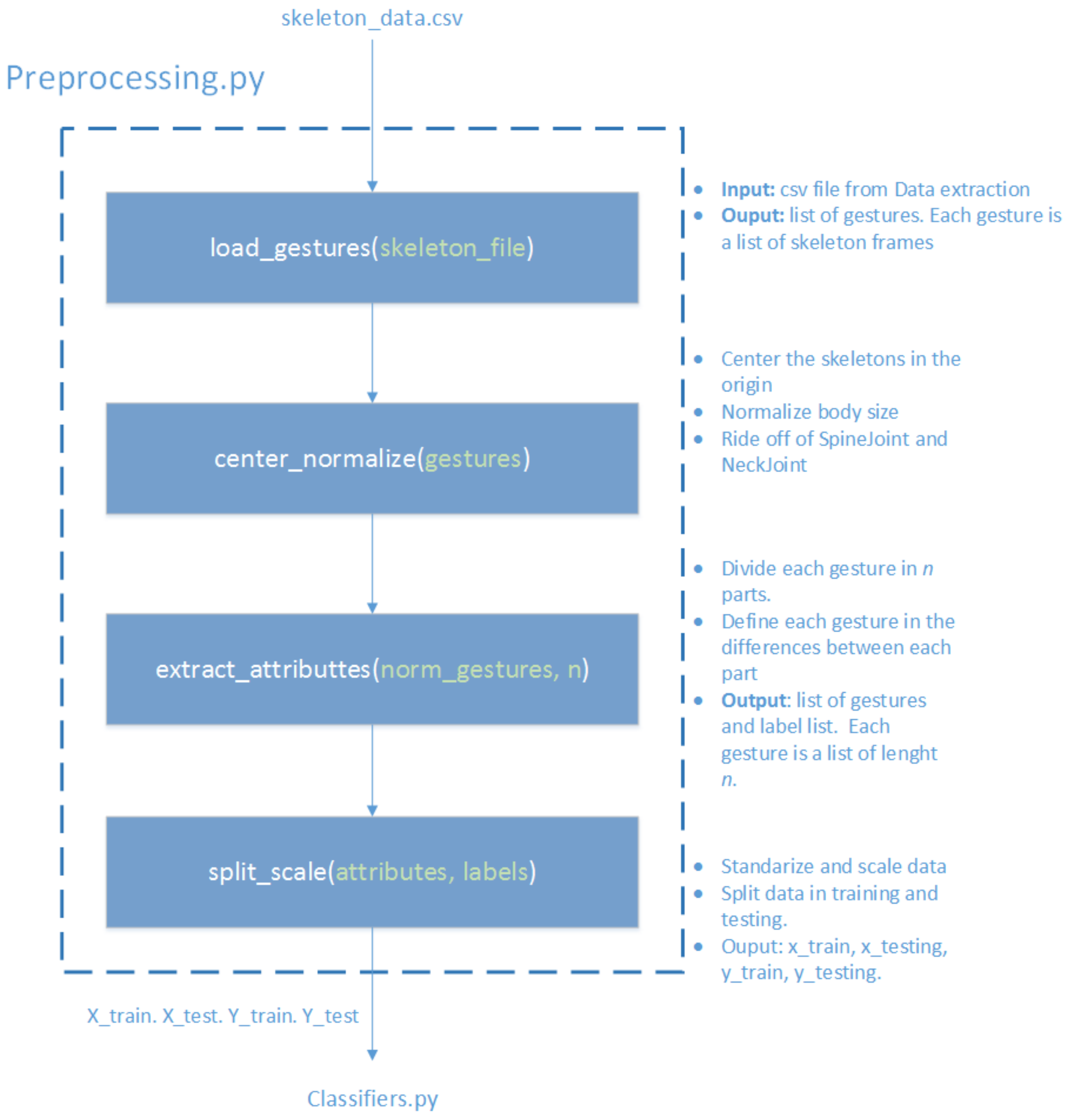

skeleton_data.csv
Preprocessing.py
load_gestures(skeleton_file)
Input: csv file from Data extraction
Ouput: list of gestures. Each gesture is a list of skeleton frames
center_normalize(gestures)
Center the skeletons in the origin
Normalize body size
Ride off of SpineJoint and NeckJoint
extract_attributtes(norm_gestures, n)
Divide each gesture in n parts.
Define each gesture in the differences between each part
Output: list of gestures and label list. Each gesture is a list of lenght n.
split_scale(attributes, labels)
Standarize and scale data
Split data in training and testing.
Ouput: x_train, x_testing, y_train, y_testing.
X_train. X_test. Y_train. Y_test
Classifiers.py

Figure 3.4: Preprocess scheme

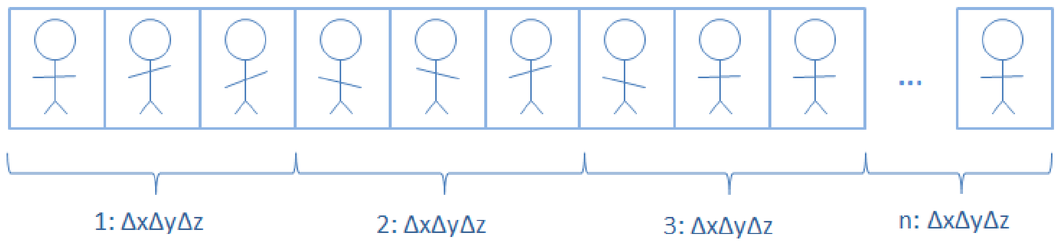


Figure 3.5: Attribute extraction

in each joint and each axis. In the Figure 3.5 we can see a graphical explanation of this method. Now the gesture is defined as:

$$Gesture = [n_{chuncks} \times 4_{joint} \times (\triangle x, \triangle y \triangle z)] + 1label$$

To define in how many parts the gesture is chunked we defined a generic value n, which we can tune to have more or less precision. The higher n the more accurate described will be the gesture. The lower n, the less amount of data the machine learning algorithms will have to handle. But the most important advantage of this step is that **the gestures are independent from the time**. Do not matter if it is executed faster or slower since it is chunked in the same number of parts. This assume that the gestures speeds are linear. Even thought that the real movements are not, it is a good approximation.

3.4.4 Standardization and split in training/testing

Standardization of datasets is a common requirement for many machine learning estimators implemented in the sckit library: they might behave badly if the individual feature do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance. For instance, if a feature has a variance that is orders of magnitude larger that others, it might dominate the learning function and make the estimator unable to learn from other features correctly as expected.

This is achieved by removing the mean of each feature and scaling by dividing non-constant features by their standard deviation.

The data is clean but there is one last step before the training phase. The algorithms used are supervised, which mean that we have to split data into training and testing data.. 80 percent of the samples are going to be used for training each system and 20 percent to test the accuracy. An important step is mingling the samples so the testing and training datasets have samples from every category. Standardization and splitting have been implemented with the *preprocessing* module of sckit library.

3.5 Training models

The remaining machine learning phases are training models, which we will explain in this section, and testing which will be explained at the Evaluation chapter. As we have said, the models we will train are **SVM, GNB, LDA and KNN.** This four models can be divided in two categories: paramteric and non-paramteric.

A parametric algorithm is computationally faster, but makes stronger assumptions about the data [4]; the algorithm may work well if the assumptions turn out to be correct, but it may perform badly if the assumptions are wrong. A common example of a parametric algorithm is linear regression. GNB and LDA fall in this category.

In contrast, a non-parametric algorithm have internal values, called hyperparameters, which could be tuned to adapt to the given data. A non-parametric algorithm,generally, is computationally slower, but makes fewer assumptions about the data. SVM and KNN falls in this category. Therefore, for non-parametric models we have to make a exhaustive search of hyperparameters that fit our data, this process explained more deeply below.

The Figure 3.6 represent the process of training and testing for both cases, parametric an non-parametric models.

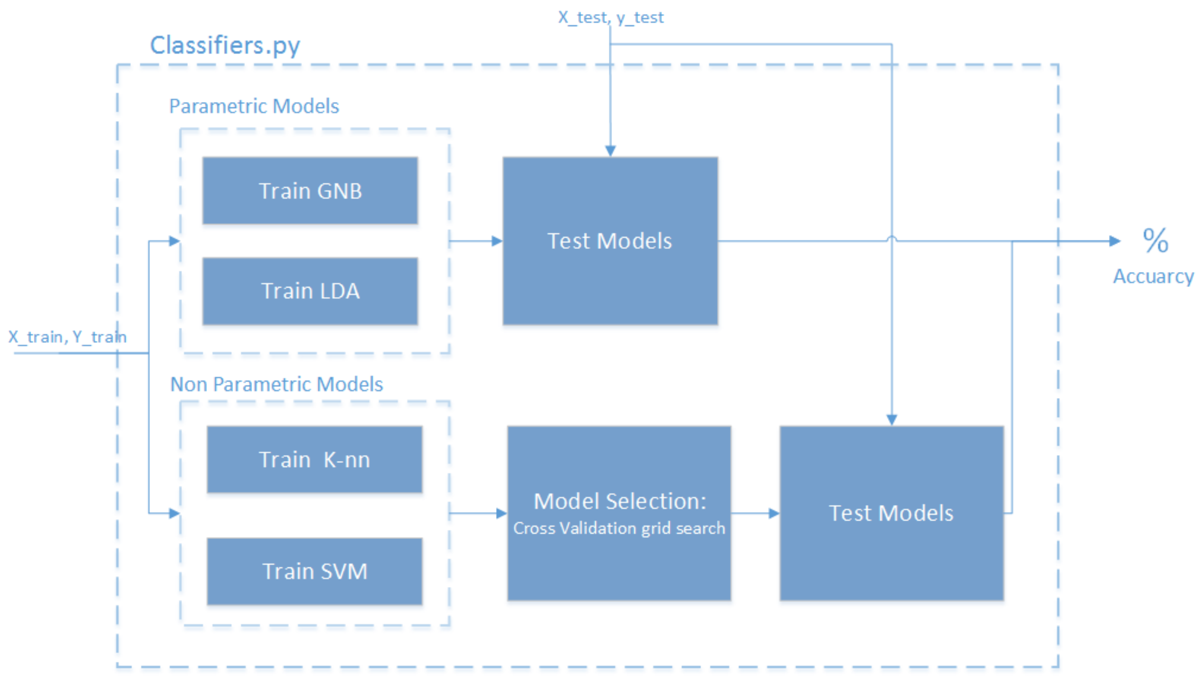


Figure 3.6: Models trained

3.5.1 Searching for estimator parameters

The hyperparameters affect the learning performance, therefore, it is necessary to tune them to find a optimal system. The process for doing this is defining a grid of parameters, implementing classifiers with all the possible combinations and evaluate which give the best performance.

Evaluate them with the test set is a mistake since there is risk of overfitting, since the testing data would "leak" into the model. A simple approach for solving this problem, is dividing the training data into real training set and **validation set**, leaving test data for the final evaluation. This validation set would be use to measure the accuracy of each hyperparameter combination tested.

However, this approach have two main issues. In one hand, we are consuming to much training data, which could be used to improve the model. On the other hand, the validation would rely on a particular random portion of the training set.

A solution for this problem is a procedure called **cross-validation**. The test set is still separated for the final evaluation, but the validation set is no longer needed. Instead, the testing data is subdivided into K parts. Then each model is trained with K-1 parts and validate with the last. Each model is trained with all the combinations of K-1 part. The performance value is given by the average.

There is several approaches for dividing the data in K parts. The one used in this project is the *Stratified k-fold*, where each partition contains approximately the same percentage of samples of each targeted class as the complete set. Also, the K value used for every validation is 5.

In the evaluation chapter we can see the grid selected for each classifier (SVM and KNN) and the results obtained.

Chapter 4

Evaluation

In this chapter we will evaluate each of the algorithm implemented. As we told in previous chapter the initial dataset was split into training and testing. Until now the testing data have been untouched in order to make bold predictions and then compare with the real label. Our original data set have 504 samples where 80%was training data and 20% testing data. In the following sections we will test:

- The **accuracy** of each algorithm over unseen data (testing data)
- If the model make **overfitting**.

Overfitting appear when the model takes to literal the training data, failing at making predictions over new data. For testing overfitting, predictions over training data will be also done

4.1 Parametric algorithms: GNB and LDA

As we said, parametric algorithms are the ones which make strong assumptions about the data distribution.

As we cans see in the Figure 4.1 the results obtained for the parametric classifiers are:

- GNB training data: 84% Testing data: 79%

REAL/PRED	Point	Zoom_in	Zoom_out	swipe_right	Swipe_left
Point	15				1
Zoom_in	2	12			
Zoom_out	2	1	20		
Swipe_right				15	10
Swipe_left	4			3	14
Precision		**Testing: 75%**		**Training: 81%**	

Table 4.1: GNB confusion matrix

REAL/PRED	Point	Zoom_in	Zoom_out	swipe_right	Swipe_left
Point	14			2	
Zoom_in		11		1	
Zoom_out		2	22		1
Swipe_right	1			22	4
Swipe_left	2				19
Precision		**Testing: 87.13%**		**Training: 93.54%**	

Table 4.2: LDA confusion matrix

- LDA training data: 93.54% Testing data: 87.13%

In the Tables **??** and **??** the rows represent the real value and the columns the prediction made. The diagonal , highlighted in blue, are the values where real and prediction matches.

None of the models make overfitting since both have very similar results over training and testing.

The confusion matrix from GNB indicate that the model have issues handling the difference between swiping right and left. LDA mistakes are fewer and more homogenous.

The better results of LDA indicate that its assumptions fits better with the dataset. The feature independence that GNB assume appear not to be very accurate. However the datasets turns to work well with a gaussian distribution assumption, since both algorithm assume it.

4.2 Non-Parametric algorithms: SVM and K-NN

The following classifiers do not make that strong assumptions of the dataset. Instead, they adapt to it, by tuning internal parameters, called hyperpa-

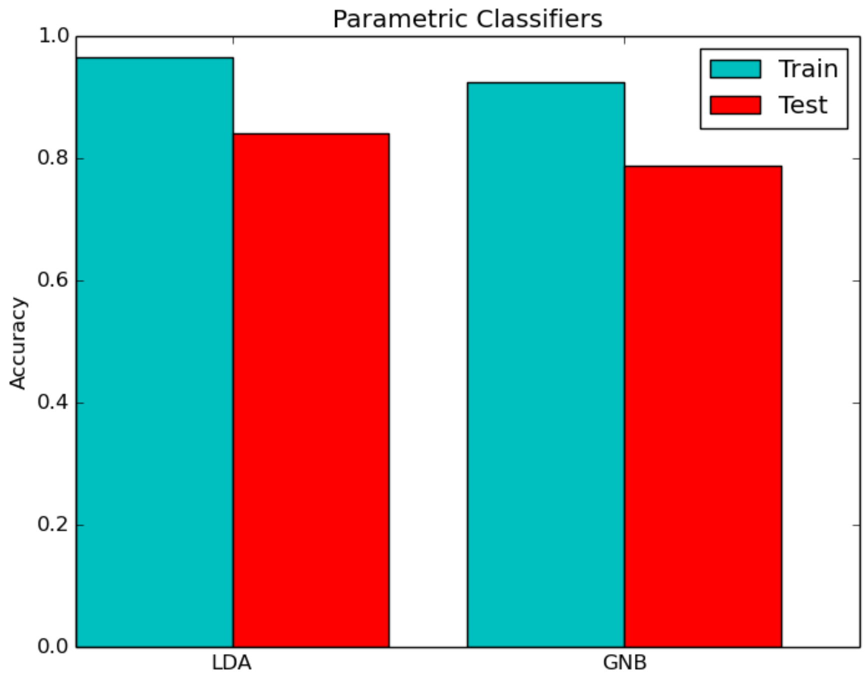


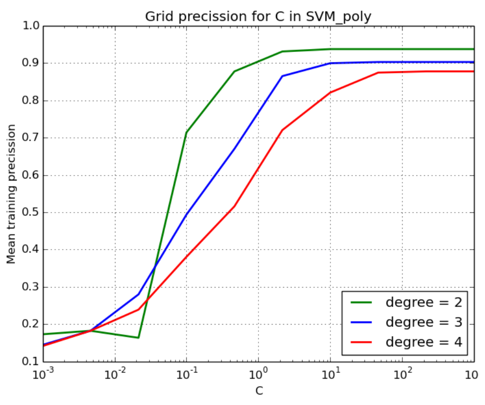
Figure 4.1: Parametric classifiers comparsion

rameters. In this section we will find the optimal hyperparameter by cross validation and evaluate the accuracy of the selected ones.

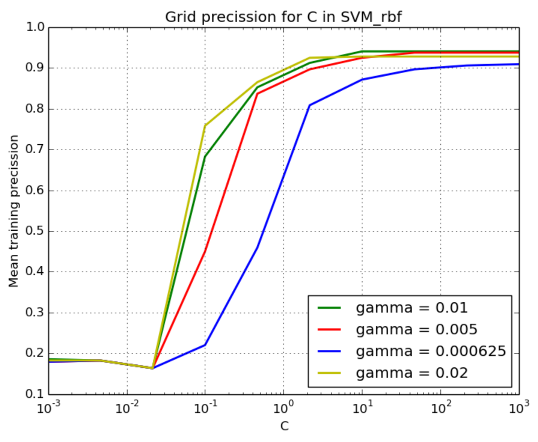
4.2.1 Grid Search for parameters selection

The method for finding the optimal parameter would be a grid search by cross validation. Training data is divided in 5 folds. Then a with a given grid of parameters, models are created and tested with all the possible combinations with the given grid. The grid for K-NN and SVM are different, since each model have different parameters to tune. Each parameter meaning is explained in the state-of-the-art chapter. Here is the grid for each algorithm:

- **K-NN** only have one parameter for tuning. K = [0 ,1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10 ,11]
- **SVM** have different Kernels and different parameters for each Kernels.



(a) Degree and C parameter search for poly SVM



(b) Gamma and C parameter search for rbf Kernel

Figure 4.2: Grid Search of different SVM kernels

This is the grid for SVM:

- Kernel = ['linear', 'poly', 'rbf']
- C = [0.001, 0.0046, 0.021, 0.1, 0.46, 2.15, 10, 46.4, 215, 1000]
- Gamma(rbf SVM): = [0.000625 0.00125 0.0025 0.005 0.01 0.02]
- Degree(poly SVM) = [2, 3, 4]

In the Figure 4.2a we can see the accuracy vs C for different *degree* values. The same in the Figure 4.2b but for different *gamma* values insted of *degree*. In the Figure 4.3 we can see the accuracy evolution for the different K over the validation data, training data and testing data. The optimal parameters are the following:

- K-nn: K = 1 with **98.87%**
- Linear SVM: C = 0.46 with **91.25%**
- Poly SVM:C = 10, degree = 2 with **96.25%**
- RBF SVM: C = 10, gamma = 0.01 with **97.5%**

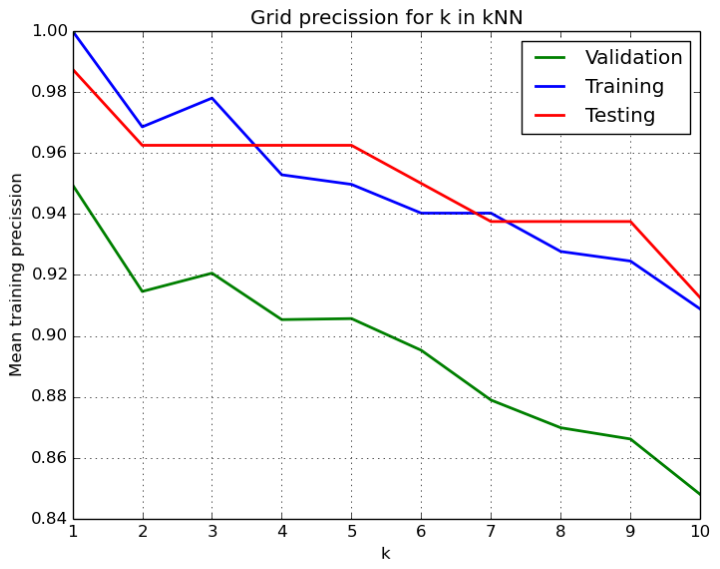


Figure 4.3: K parameter search for K-NN

4.2.2 Accuracy test for models selected

The selected models are 1-NN and SVM with a RBF Kernel. In the Tables 4.3 and 4.4 we can see the confusion matrix for each model. We get very good accuracies for both algorithms and no signs of overfitting.

REAL/PRED	Point	Zoom_in	Zoom_out	swipe_right	Swipe_left
Point	15				1
Zoom_in		12			
Zoom_out			25		
Swipe_right				26	1
Swipe_left					21
Precision	**Testing: 98.02%**			**Training: 100%**	

Table 4.3: SVM(Kenrel=RBF, C = 10, gamma= 0.01) confusion matrix

4.3 Parametric vs non-parametric

Here we compare all the algorithms implemented, as we can see in the Table 4.5. Parametric classifiers have perform better since they have adapted to the data. Finally the best result is for the K-NN with a **99.01**% of matches and no overfitting.

REAL/PRED	Point	Zoom_in	Zoom_out	swipe_right	Swipe_left
Point	16				
Zoom_in		12			
Zoom_out			25		
Swipe_right				26	1
Swipe_left					21
Precision	**Testing: 99.01%**			**Training: 100%**	

Table 4.4: 1-NN confusion matrix

	Accuracy			
	Parametric		**Parametric**	
	GNB	LDA	SVM	KNN
Testing(%)	75	87.13	98.02	**99.01**
Training(%)	81	93.54	100	100

Table 4.5: Accuracy comparison between the different models

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Once finished the development of the project, we will focus on whether the objectives specified at the introduction of the document have been fulfilled. Remembering, our main objective was recognize gestures with machine learning techniques, creating a flexible platform. This involved two phases: (1) creating a platform for recording and extracting data from the users and (2) implementing and testing several machine learning algorithm to find the one that better fits our dataset.

For the first part we have successfully created a recording program through Kinect and the official SDK. The program allow us to visualize in real time the data while recording and limit the gesture by giving the chance to start and stop recording at anytime. We have created a dataset of 5 gestures for the gesture recognition part. However this program allow us to record the gestures that the user would want. Hence, we maintain our reusability and flexibility objective.

For the second part, gesture recognition, we have implemented up to 4 different machine learning algorithm: K-NN, SVM, LDA and GNB. We have implemented both, parametric and non-parametric algorithms and observed how the non-parametric algorithms, with previous optimizations, turned to have better accuracy. The best algorithm have been K-NN with a **99.01%** of accuracy on the testing dataset and 100% over the training dataset which indicate no overfitting. This is a very good results that are enough to be implemented in a real product, and we can conclude that we have fulfil the initial goals.

5.2 Future work

For future works we consider the following ideas:

5.2.1 Automatic gesture spotting

The platform created have been trained with 5 gestures. In the training dataset all the samples is one of this gestures and when making predictions the gesture with higher probability is the chosen one. However, if we want to make predictions of real time data there would be samples which are not related with any of the selected gestures. A technique to differentiate when a gesture is performed or not is necessary. In the open source tool GRT from Nick Gillian[5] they propose the following solutions for gesture auto-spotting.

- The user could press a special gesture button as they make a gesture to indicate a gesture is being performed.
- We could create a special null gesture class and teach the classifier to recognize more null gestures (i.e. general movements) from valid gestures.
- We could try and estimate a threshold value that rejects samples which are unlikely to belong to any of the classes in our model.

The first option would not be very friendly user, while the second option look unfeasible since the amount of general movements for our project are

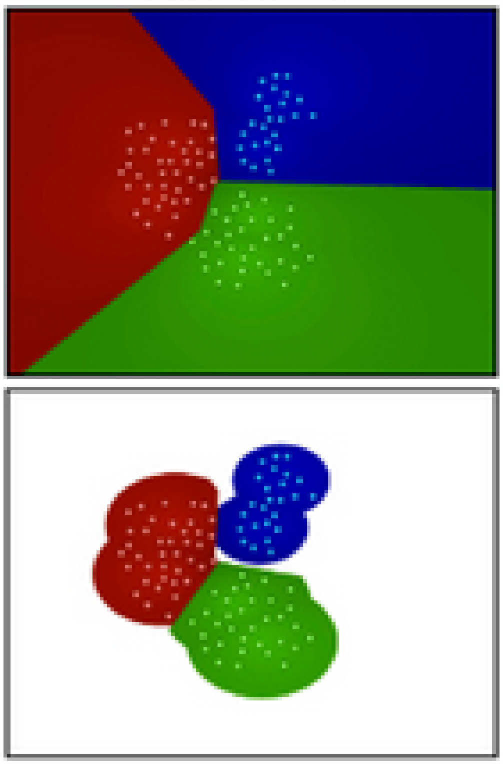


Figure 5.1: Auto-spotting

almost infinite. The third option look much more adequate to our case. This solution would is graphically represented in the Figure 5.1.

5.2.2 Create a application

The gesture recognition platform can be used as a tool to create a application. There is many areas where gesture recognition are useful, one example is the one that indicate our dataset. The gestures can be used for controlling a power point presentation. When the gesture platform generate a match it can associate with operations such as swiping the slide, making zoom, selecting.

Chapter 6

Project Management and Budget

In this chapter we will analyze how the project have been managed and resources used. This way we can get a overview on how the initial planning was made and how it changed with the time. Then we can evaluate which have been the main difficulties we have found on the way.

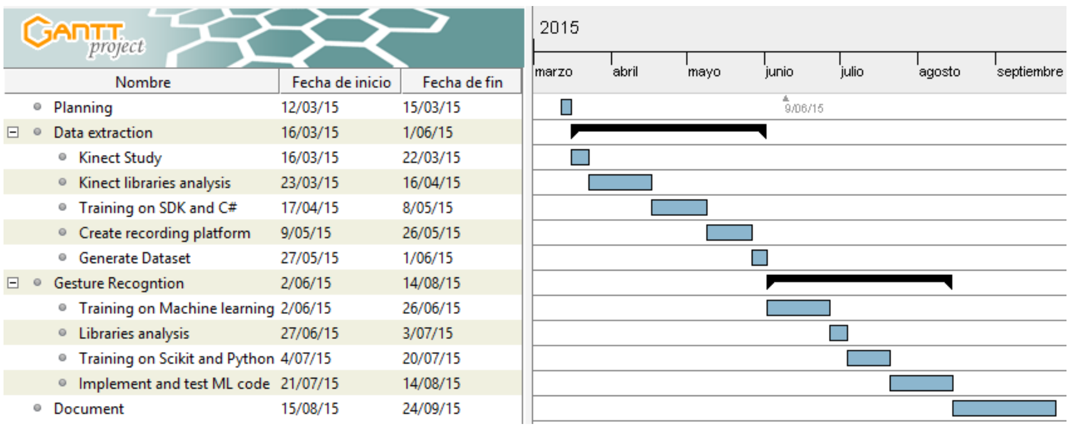
6.1 Project Management

6.1.1 Initial planning

In the Gantt diagram from the Figure 6.1 we can see which one was the initial planning for the project. In the planning weekends have also being included. In total it sum up to 172 days and considering 3 hours per day it sum up to 566 hours.

6.1.2 Final planning

In the Figure 6.2 we can see the actual planning that was followed during the realization of the project. If we compare it with the initial planning Ganntt chart, we observe that the *Kinect libraries analysis* was overex-



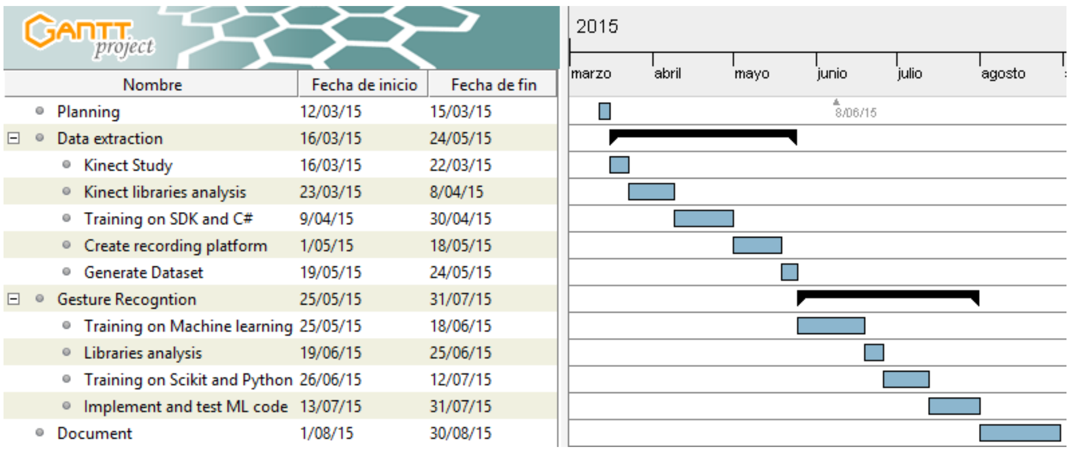
Nombre	Fecha de inicio	Fecha de fin
Planning	12/03/15	15/03/15
Data extraction	16/03/15	1/06/15
Kinect Study	16/03/15	22/03/15
Kinect libraries analysis	23/03/15	16/04/15
Training on SDK and C#	17/04/15	8/05/15
Create recording platform	9/05/15	26/05/15
Generate Dataset	27/05/15	1/06/15
Gesture Recogntion	2/06/15	14/08/15
Training on Machine learning	2/06/15	26/06/15
Libraries analysis	27/06/15	3/07/15
Training on Scikit and Python	4/07/15	20/07/15
Implement and test ML code	21/07/15	14/08/15
Document	15/08/15	24/09/15

Figure 6.1: Initial Grantt chart

tended eight days, the *Implementation and test of ML code* was overextended six days and the *Documentation* phase was overextended ten days. We can observe that the ending milestone that both tutor and student set was missed, having a total delay of twenty four days. When designing the planning for the project this possibility was taken into account so the consequences (due to the time until the official hand) still allow to finish the project on time.

It is interesting to discuss the cause of some of the delays to consider in future projects. During the Kinect libraries analysis, the main idea was making the whole project with open source tools. However, after trying to use OpenNI, we found a big lack of documentation, which turned in complications on driver installations and poor features from the available tools. Was not until we found that we realize that PrimeSense, the main supporter of OpenNI, was bought by Apple which closed the whole project. This explained the lack of documentation and development since 2013 and we decided to use the Official Mircosft SDK.

The delay in the documentation phase was, especially, for the using of the text editor Latex which was not planned. The lack of experience in this new tool needed additional learning, which turned into slower writing.



Nombre	Fecha de inicio	Fecha de fin
Planning	12/03/15	15/03/15
Data extraction	16/03/15	24/05/15
Kinect Study	16/03/15	22/03/15
Kinect libraries analysis	23/03/15	8/04/15
Training on SDK and C#	9/04/15	30/04/15
Create recording platform	1/05/15	18/05/15
Generate Dataset	19/05/15	24/05/15
Gesture Recogntion	25/05/15	31/07/15
Training on Machine learning	25/05/15	18/06/15
Libraries analysis	19/06/15	25/06/15
Training on Scikit and Python	26/06/15	12/07/15
Implement and test ML code	13/07/15	31/07/15
Document	1/08/15	30/08/15

Figure 6.2: Final Gantt chart

6.2 Budget

Once that the planning and duration is set, we can detail the budget related to the project. We will include the direct cost (personal, materials, etc.), as well as the indirect cost (office, security, paper, etc). As we have seen in the project management, the time invested in the project have been 7 months. The following calculations will be considering this.

6.2.1 Hardware

For the project have been needed a computer as well as the Kinect sensor. The calculations in the Table 6.1 have been made taking into account the warranty of each piece of hardware.

Description	Cost (EUR)	%use for the project	Time used (months)	Lifespan (months)	Chargeable cost
Laptop	599 EUR	100%	7	24	174.7 EUR
Kinect XBOX 360	70 EUR	100%	7	24	20.4 EUR
				Total	**195.1 EUR**

Table 6.1: Hardware chargeable cost

Description	License	Cost/License (EUR)	Cost
Kinect for Windows SDK 1.8	1	0.00	0.00
Microsoft Visual Studio 2013	1	0.00	0.00
Kinect for Windows SDK 1.8	1	0.00	0.00
Latex editor 1.8	1	0.00	0.00
GranttProject SDK 1.8	1	0.00	0.00
		Total	**0.00 EUR**

Table 6.2: Software cost

6.2.2 Software

Most of the used software have been open source with free license. Visual Studio 2013 have been used under student license, so there is not additional charges. It is detailed in the Table 6.2

6.2.3 Personal

To complete this project one junior engineer have been required (the student). According to a study made by the JobTonic webpage the average salary for a junior engineer is 18000 EUR per year. This is the value taking in account for the calculations in the Table 6.3.

Position	Annual salary	Months working	Cost
Junior Engineer	18000 EUR	7	Cost 10500 EUR
		Total	Cost **10500 EUR**

Table 6.3: Personal cost

6.2.4 Total

Indirect costs are those that are derived from the operation of an enterprise and not linked to only one product(office, security,etc.). According to Rodrigo and Berges [9], indirect costs in most Spanish university investigation projects are calculated as a total percentage of the direct costs. The percentage chosen have been 10%.

In the Table 6.4 we can observe that the budget is of **14230.23 EUR**, considering a taxation of 21%.

Asset	**Cost**
Hardware	191.5 EUR
Software	0.0 EUR
Personal	10500 EUR
Indirect cost(10%)	1069.5 EUR
Total cost without tax	11760.5 EUR
Tax (21%)	2469.73
Total cost (Tax included)	**14230.23 EUR**

Table 6.4: Total cost

Bibliography

[1] Elise Ackerman. Apple reportedly set to buy 3d sensing company as it prepares for new generation of devices that can 'see'. *Forbes*, 2013.

[2] L. Schiaratura B. Rimé. Gesture and speech. fundamentals of nonverbal behaviour (pp 239-281). 1991.

[3] C. M. Bishop. Pattern recognition and machine learning. 2006.

[4] S. Geisser and W.M. Johnso. Modes of parametric statistical inference. 2006.

[5] N. Gillian and J. Paradiso. The gesture recognition toolkit. *Journal of Machine Learning Research*, 15, 2014.

[6] W. Hinchman. Kinect for windows sdk beta vs. openni. 2011.

[7] Fernando Fernández andCristina Suárez-Mejías José Carlos Gonzaléz, José Carlos Pulido. Planning, execution and monitoring of physical rehabilitation therapies with a robotic architecture. 2015.

[8] J. C. R. Licklider. Man-computer symbiosis. HFE-1:4–11, 1960.

[9] I. Rodrigo Martínez and C. Villarroya Berges. Costes indirectos de la investigación bajo contrato en la universidad española. 2001.

[10] Gonzalo Mariscal Óscar Marban and Javier Segovia. A data mining knowledge discovery process model. 2009.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Alfredo Teyseyre Rodrigo Ibañez, Álvaro Soria and Marcelo Campo. Easy gesture recognition for kinect. *Advances in Engineering Software*, 76, 2013.

[13] Peter Russell, Stuart; Norvig. Artificial intelligence: A modern approach. 2003.