Escuela
Politécnica
Superior

Hand gesture recognition for human-computer interaction using low-cost RGB-D sensors

Bachelor's thesis

Bachelor's Degree in Computer Engineering

Bachelor's Thesis

Author:
Sergiu Ovidiu Oprea

Supervisors:
José García Rodríguez
Sergio Orts Escolano

Universitat d'Alacant
Universidad de Alicante

September 2015

UNIVERSITY OF ALICANTE

Bachelor's Degree in Computer Engineering

Bachelor's Thesis

Hand gesture recognition for human-computer interaction using low-cost RGB-D sensors

Author

Sergiu Ovidiu Oprea

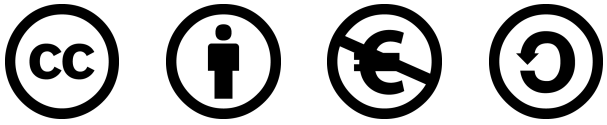
Supervisors

José García Rodríguez, PhD.

Sergio Orts Escolano, PhD.

Department of Computer Technology (DTIC)

Alicante, September 9, 2015



Logic will get you from A to B.
Imagination will take you everywhere.
Albert Einstein

Acknowledgments

First of all, I hope you have fun with this acknowledgment section. It will be the last and only funny text that you will read of the whole document. Look into my eyes, promise me that you won't insult me when you get to the last page of this document, where the time loss feeling seized of your own body and incited you to hang yourself with the belt of the living room lamp. Thank you, my apologies.

Despite the cruelty of the introduction, I would like to express my sincere gratitude to my supervisors, Jose and Sergio. Your persistence, perseverance, patience and enthusiasm made it all possible. It was a pleasure working with you. Thanks again for everything.

I would also like to thank APSA association for its direct collaboration and for the provided suggestions to this project, to DTIC department specially to the I2RC research group and finally, to DCCIA department for the research grant founded by MECD of Spain.

I would also like to express my sincere gratitude to my parents and grandfather for helping me to survive to all the stress from these last four years. They have supported me, have educated me and have always loved me. This thesis is for you.

Thanks to all my colleagues and friends for all the funny experiences we have shared. Thanks Albert, Javi, Pablo, Ginés, Manu, Cayetano, Víctor, Alex, Jorge and Brayan. You made it worth every single day. Also thanks to: Vicente, Miguel, Marcelo, Joan Carles, Pablo, Jose Manuel, Fran and Joselu. Special thanks to Fraser for being so crazy. They have always been there.

I would also like to thank Anaid, for putting up with me almost all the career. I imagine that if it has been difficult for me, must have been difficult for her too.

We shouldn't forget of all those crazy girls (without offending or generalize) who have wasted part of my precious time dedicated to this project.

Thanks to *Foster's Hollywood* for their *Bacon Cheese Fries* and *Montreal Ribs*. Without these necessary nutrients, this thesis wouldn't have been possible.

It's 5 am, so I should go to sleep. Have a good day!

Abstract

In recent years, research in gesture recognition for human-computer interaction has increased significantly. There are many references to projects and papers with relevant content regarding this research topic. The gestures made with the hands and arms have received more attention as they are considered parts of the body with a high degree of freedom (DOF), flexibility and high-expressiveness. This project will be focused to the gesture recognition made with hands and arms in favorable conditions and also in situations with occlusions in the movements and under demanding time constraints.

In this project, the design and implementation of a gesture recognition system on mobile robotic devices with moderate consumption is proposed, allowing its use in mobile robotics systems, intelligent environments or ambient assisted living applications.

The data acquisition system is based on the processed information flows of color, depth and skeleton data obtained from RGB-D low-cost sensors such as *Microsoft Kinect* and *Primesense Carmine*.

The range of algorithms and applications will cover several levels, from the acquisition and improvement or filtering of the input data, through the segmentation and characterization of relevant areas in the scene and ending with the detection and recognition of static and dynamic gestures. The results can be applied to systems of human-computer interaction for the navigation control of a robot or gestural interaction.

Resumen

El reconocimiento de gestos para la interacción hombre-máquina es un tema de investigación muy activo y estudiado en la actualidad, existiendo numerosas referencias a proyectos y publicaciones con contenido relevante. Los gestos realizados con las manos y brazos han recibido más atención ya que se consideran partes del cuerpo con un elevado grado de libertad (DOF) y flexibilidad, además de alta expresividad. Enfocaremos este proyecto al reconocimiento de gestos realizados con las manos y brazos no solamente en situaciones favorables sino también en condiciones con oclusiones en los movimientos y bajo restricciones temporales.

En este proyecto se propone el diseño y la implementación de un sistema de reconocimiento de gestos sobre dispositivos móviles de consumo moderado permitiendo su utilización en aplicaciones de robótica móvil, de ambientes inteligentes o de vida asistida por el entorno.

El sistema de adquisición de datos se basa en el procesamiento de flujos de información de color y profundidad (RGB-D), además de información del esqueleto del usuario en cuestión obtenida a partir de sensores de bajo coste como *Microsoft Kinect* y *Primesense Carmine*.

El abanico de algoritmos y aplicaciones cubrirá varios niveles, desde la adquisición y mejora o filtrado de los datos de entrada, pasando por la segmentación y caracterización de zonas de interés en la escena y terminando por la detección y el reconocimiento de gestos estáticos y dinámicos. Los resultados pueden ser de aplicación a sistemas de interacción hombre-máquina para el control de la navegación de un robot o interacción mediante ordenes gestuales.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

This first chapter introduces the main topic of this work. It is organized in six different sections: Section 1.1 sets up the framework for the activities performed during this project, Section 1.2 introduces the motivation of this work, Section 1.3 elaborates a state of the art of existing 3D dynamic and static gesture recognition systems, Section 1.4 lays down the proposal developed in this work, Section 1.5 presents the main and specific goals of this project. Finally, Section 1.6 details the structure of this document.

1.1 Outline

In recent years, and with the advent of low cost 3D sensors, research in posture and gesture recognition for human-computer interaction (HCI) has increased significantly. Posture and gesture recognition are relevant topics in the fields of artificial intelligence, computer vision and image processing, and can be focused to specific parts of the body such as arms, hands, eyes or directly to the whole body. Nevertheless, hand and arms gestures have received more attention because of its high degree of freedom (DOF) and flexibility. Hand is the most effective, general-purpose interaction tool due to its dexterous functionality in communication and manipulation [1]. This fact justifies the effort devoted to create easier to use interfaces by directly employing natural communication and manipulation skills of humans.

This bachelor thesis is focused on the theoretical and practical research on hand posture and gesture recognition in cluttered environments under demanding time constraints for people with acquired brain damage or other disability. The project comprises the study of 3D hand posture and gesture recognition systems combined with 3D data acquisition by means of low-cost sensors such as *Microsoft Kinect v1 and v2* devices.

The main goal of this project is to design and implement an efficient and accurate solution for 3D gesture recognition able to run on real time and at the same time, integrated on a low cost mobile robotic platform.

This work has been performed under the frame of the following national project: *SIRMAVED: Development of a comprehensive robotic system for monitoring and interaction for people with acquired brain damage and dependent people*, ID code (DPI2013-40534-R), funded by the *Ministerio de Economía y Competitividad (MEC)* of Spain with professors José García-Rodríguez and Miguel Ángel Cazorla-Quevedo from the University of Alicante as head researchers.

Moreover, part of this work was done during an eight month collaboration in the *Department of Computer Science and Artificial Intelligence (DCCIA)* at the University of Alicante. This collaboration was funded by the *Ministerio de Educación, Cultura y Deporte (MECD)* of Spain by means of a research grant for the project: *Implementation of 3D vision algorithms under time constraints: human-computer interaction and object recognition applications*. The work carried out was led and supervised by the professors José García-Rodríguez and postdoctoral researcher Sergio Orts-Escolano from the *Industrial Informatics and Computer Networks* research group.

1.2 Motivation

This document presents the results of the work carried out for finishing the *Bachelor's Degree in Computer Engineering*, taken between the years 2011 and 2015 at the *University of Alicante*. The main motivation of this work arises from the collaboration with the *Department of Computer Technology* in research tasks related to

computer vision, high performance computing and the *SIRMAVED* project.

The main goal of the previously mentioned *SIRMAVED* project is the development of a robotic system for monitoring dependent people or persons with acquired brain damage and interacting with them to perform common daily tasks like recognition and manipulation of small sized objects. In order to do that, the integration of multiple technologies is required; particularly, an human-computer interaction system (HCI) in this case, between patient and robot, is needed. In that sense, we have to design and develop an on-board 3D posture and gesture recognition system which must be able to operate on cluttered environments considering occlusions in patient movements. The system must meet the low power, small size and real-time requirements.

At the same time, the system was designed in order to recognize Schaeffer language used by children and people with intellectual or cognitive disabilities. A direct collaboration with *APSA* (*Pro Mentally Handicapped Association from Alicante*) was performed.

1.3 Related works

In recent years, and with the advent of low cost 3D sensors, research in posture and gesture recognition for human-computer interaction (HCI) has increased significantly. The existing technologies for HCI become a bottleneck in the effective utilization of the available information flow [2].

In Computer Graphics, the interaction consists of the direct manipulation of graphic objects such as icons and windows using a pointing device. Nevertheless, the particular case of interaction with 3D objects, is still a complicated situation in which mechanical devices such as keyboard and mouse are incompatible. The two degrees of freedom (DOFs) of the mouse cannot properly emulate the three dimensions of space. In this case, the use of hand gesture recognition systems provides an attractive and natural alternative to the cumbersome interface devices for human-computer interaction. Using hands as a device can help people communicate with computers in a more intuitive way [2].

Posture and gesture recognition can be focused to specific parts of the body such as arms, hands, head, eyes or directly to the whole body. Nevertheless, the hand and arms gestures have received more attention because they are body parts with a high degree of freedom (DOF) and flexibility (See 1.3.2.1). The hand is the most effective, general-purpose interaction tool due to its dexterous functionality in communication and manipulation [1]. Hand movements are a mean of non-verbal communication, ranging from simple actions (e.g. pointing at objects) to more complex ones (e.g. expressing feelings or communicating with others) [2]. Therefore, this document will deal with both static and dynamic gestures made by hands and arms.

In everyday life, people use gestures to enrich verbal communication (due to its high expressiveness) or in this particular case, for human-computer interaction with the purpose of transmitting information or orders. This fact justifies the effort devoted to create easier to use interfaces by directly employing natural communication and manipulation skills of humans. Adopting direct sensing in HCI will allow the deployment of a wide range of applications in more sophisticated computing environments such as Virtual Environments (VEs) or Augmented Reality (AR) systems.

1.3.1 Gesture definition

According to [3], a gesture can be defined as a physical movement of the hands, arms, face and body with the intent to convey information or meaning. The word gesture is used to group a series of different interactive or communicative human movements, especially of the hands and arms [2]. A gesture is a movement which shows or symbolizes something and contains a message. In contrast, it would simply be a functional movement lacking any meaning. Waving a good-bye is a gesture. Pressing a key on a keyboard is not a gesture because the motion of a finger on its way to hitting a key is neither observed nor significant. All that matters is which key was pressed. At the same time, writing down words on a sheet of paper is a pure functional movement. The words contain the information,

not the movement of the hand. In this document we will talk about two different types of gestures: static and dynamic gestures. Static hand gestures are defined as orientation and position of hands in space during an amount of time without any movement [4]. Meanwhile, dynamic gestures are defined as orientation and position of hands in space with movement during an amount of time [4]. The main difference between both gestures is the presence of movement.

1.3.2 Hand gestures

Due to the ability of human hands to acquire a huge number of clearly discernible configurations, the category of hand gestures is the most numerous. According to different application scenarios, hand gestures can be classified into several categories such as conversational, controlling, manipulative and communicative gestures [2]. Sign language (e.g. Schaeffer language) is an important case of communicative gestures. They can be a good way to help disabled people to interact with computers. On the other hand, analyzing pointing gestures, a person with reduced mobility (PRM) will be able to locate virtual objects or even the real objects of a scenario, for example, his own room. Generally gestures can be classified into static or dynamic, hand posture or gesture, respectively.

Although the term hand gesture and posture refers to the same meaning, there are some differences between them. Hand posture is defined as a static pose, holding the hand with specific pose is a posture, for example a victory sign, pointing, and thumbs up. These static gestures can be very complex because of the high degree of freedom (DOF) and flexibility of hand fingers. At the same time, gesture can be defined as a dynamic movement, such as waving goodbye. A complex gesture is one that includes three things as mentioned by Mitra [3] which are finger and wrist movement and changes in hand's position and orientation (occlusions and movement speed). In conclusion, hand gestures fall into two categories, namely static and dynamic. Some hand gestures also have both static and dynamic elements, as in sign languages: American Sign Language signs [5] and Schaeffer language [6]. In this document we will focus mainly on

dynamic gestures. Nevertheless, we will also handle static gestures due to the lack of movement in many implemented gestures.

1.3.2.1 Hand morphology and kinematic model

The human hand consists of 27 bones, 8 of which are located in the wrist. The other 19 constitute the palm and fingers as shown in Figure 1.1a. The bones in the skeleton form a system of rigid bodies connected together by joints with one or more degrees of freedom for rotation [1]. The name of the joints between the bones are chosen according to their location on the hand as metacarpophalangeal (MCP) (i.e. joining fingers to the palm), interphalangeal (IP) (i.e. joining finger segments) and carpometacarpal (CMC) (i.e. connecting the metacarpal bones to the wrist). All five MCP joints are described as saddle joints with two DOF: abduction/adduction (i.e. spreading fingers apart) in the plane defined by the palm, and flexion/extension. The nine IP joints have only one DOF, flexion/extension. The CMC of the index and middle fingers are static while the CMC of the pinky and the ring finger have limited motion capability. The CMC of the thumb, also called trapeziometacarpal (TM), is the most difficult to model because TM joint has two non-orthogonal and non-intersecting rotation axes [7]. A 27 DOF model used in many studies is shown in Figure 1.1b.

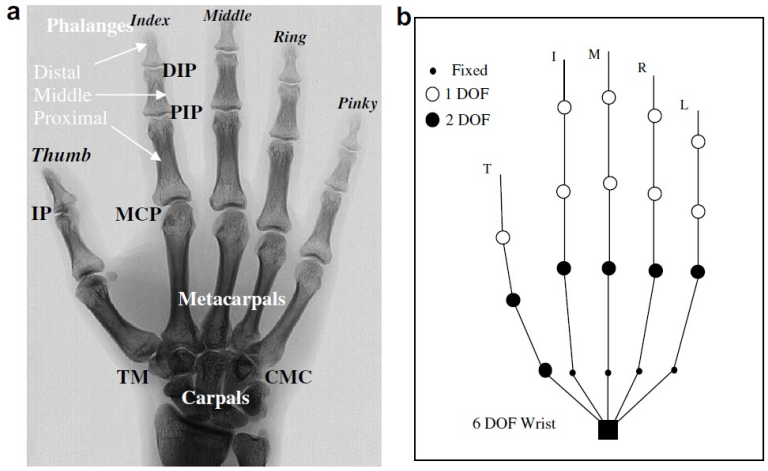


Figure 1.1: Skeletal hand model: (a) Hand anatomy, (b) the kinematic model (Figure reproduced from [1])

1.3.3 Hand gesture recognition approaches

There are two main groups of techniques in hand gesture recognition: ***glove-based and computer vision-based approaches***.

By one hand, glove-based approach is based on the physical interaction of the user with a device, such as a data glove, accelerometers, multi-touch screen and other sensors. By the other hand, computer vision-based approach represents a promising alternative because of its potential to provide more natural and non-contact interaction. Using RGB cameras and depth sensors beside computer vision algorithms and image processing techniques, the gesture recognition process will be a real challenge for researchers.

Nevertheless, both approaches have their pros and cons. The glove-based systems can be uncomfortable for users since they require physical contact, even so, they have a verge over the accuracy of recognition and less complexity of implementation. Vision-based systems are user friendly but suffer from configuration complexity and occlusion problems. We will delve into both techniques in the following sections.

1.3.3.1 Glove-based gesture recognition

The history of hand gesture recognition for human-computer interaction started with the invention of glove-based control interfaces. This technology gradually improved with the development of accurate accelerometers, infrared cameras and even fiberoptic bend-sensors (optical goniometers) [8]. Over past 25 years, this evolution has resulted in many successful products that offer total wireless connection. Despite all the progress in the glove-based techniques and at the same time, taking into account the techniques based on computer vision, currently, the only technology that satisfies the advanced requirements of hand-based input for HCI is glove-based sensing [9]. The most effective tools for capturing hand motion are electro-mechanical or magnetic sensing devices (data gloves) [10].

Data glove in essence is a wired interface with certain tactile or other sensory

units that were attached to the fingers or joints of the glove, worn by user [8]. The tactile switches, goniometers or optical sensors measure the bending resistance of different hand joints providing crude measurements. In this way, we could easily determine if a hand is open or closed or some finger joints are straight or bent. The results were mapped to unique gestures and were interpreted by a computer. The main advantage of this devices was that there was no requirement for any kind of pre-processing. With very limited processing power on computer back in 1990s, these systems showed great promise despite the limited hand motion.

These devices are worn on the hand to measure the location of the hand and finger joint angles. With a complex calibration and setup procedures, measurements will be accurate and precise with quite small error bounds that are known beforehand. However, they have several drawbacks [1] in terms of casual use as they are very expensive and the calibration process is complicated. At the same time, data-glove based systems involve the user to carry in hands, cumbersome and complex devices (See Figure 1.2) that hinder the naturalness of hand motion. Moreover, in most cases, these devices are connected to the computer. Thus, the natural level of user's interaction with the computer is reduced, limiting the movement and freedom of hand to make a gesture.

By looking at the evolution of data gloves, there were two distinct categories emerged over the years [8]:

- **Active data glove** - consisted of few or variety of sensors on the glove to measure flexing of joints or acceleration and had a communication path to the host device using wired or wireless technology (See 1.2a).

- **Passive data glove** - consisted only of markers or colours for finger detection by an external device such as a camera. The glove did not have any sensors onboard (See 1.2b).

In the case of active data gloves the measurements are provided by a variety of different sensors placed in the same glove as we can see in the Figure 1.2a. Moreover, in the case of passive data gloves (colored gloves, Figure 1.2b), an external device as a RGB camera is required in order to apply a color segmentation pro-

(a) MIT *Acceleglove* glove-based system with its multiple sensors (Figure reproduced from [11]).

(b) MIT glove design consisting of large color patches (Figure reproduced from [12]).

Figure 1.2: Left: Active data glove with multiple sensors. Right: Passive data glove.

cess. This technique is considered a computer vision-based gesture recognition approach. We will delve into CV based techniques in the section 1.3.3.2.

1.3.3.2 Computer vision-based gesture recognition

With the advent of computer vision (CV) techniques, data gloves were left in the background. In contrast to non-vision based recognition, vision based approaches represent a promising alternative because of its potential to provide more natural, unencumbered, non-contact interaction [1]. Vision-based approaches are difficult to implement in a satisfactory manner. Researchers have to overcome several restrictions regarding to hand morphology, modeling and uncontrolled environments (See sections 1.3.2.1, 1.3.4) For more differences between these two approaches, see table 1.1.

The main challenge of vision-based hand gesture recognition is to cope with the large variety of gestures [4]. The recognition of a hand gesture involves handling a considerable number of degrees of freedom (DoF), huge variability of the 2D appearance depending on the camera view point, different silhouette scales, self-occlusions and hand-hand occlusions and many resolutions for the temporal

Criterion	Contact-devices	Vision-devices
User cooperation	Yes	No
User intrusive	Yes	No
Precise	Yes/No	No/Yes
Flexible to configure	Yes	No
Flexible to use	No	Yes
Occlusion problem	No	Yes
Health issues	Yes	No

Table 1.1: Comparison between contact-devices and vision-devices [4]

dimension, such as, variability of the gesture speed. A balance between accuracy, performance, usefulness, real-time constraint, robustness, scalability, user-independence and cost of the solution is required. We mean by real-time performance, the analysis of the input images at the frame rate of the input video in order to give the user instant feedback of the recognized hand gesture. Robustness plays an important role in recognizing different hand gestures successfully under lighting condition variation, cluttered backgrounds and against in-plane and out-of-plane image rotations. Scalability involves the requirement to manage a large gesture vocabulary which can be included with a small number of primitives. User-independence creates the environment where the system can be handled by different users rather than specific user and should further recognize gestures performed by humans of different age (kids/adults) or skin color.

Some researchers have proposed the use of markers that are attached to the skin or colored gloves (See Figure 1.2b) to simplify the segmentation and recognition process [13, 14, 15, 12].

Using a camera, the system then computed motion trajectories and used them to determine the start and the end position of the gesture. The aim of these approaches has been to develop a low cost system against the active data gloves with much more flexibility and very low computation requirements.

As the marker approaches appear to be intrusive and not natural, marker-

less motion capture techniques seems to be more efficient and more natural. The first instance of a hand recognition system that totally relied on computer vision without markers was reported by Rehg and Kanade in 1993 [16]. The system was called DigitEyes and the absence of the requirement to wear any glove or colored markers was its main contribution. Rehg and Kanade presented tracking results for a human hand model which resulted in a highly articulated model (27 DOFs). The success of this CV approach was mainly due to their ability to extract simple and useful features from human hand. In order to avoid occlusion across complicated backgrounds, they used two cameras.

Darrell and Pentland (1993) developed a method for learning, tracking and recognizing human gestures using a view-based approach to model articulated objects [17]. In their approach, objects were represented using sets of view models and stereotypical space-time patterns, such as gestures, were then matched to stored gesture patterns using dynamic time warping (DTW).

These first CV-based approaches always suffered of the low camera resolution and of the inadequate computing power that directly preclude the operation of the system in real time and limit the overall progress of gesture recognition systems.

Utsumi et al. first used a stereo rig in order to determine the Center of Gravity (COG) of the palm and calculate fingertips location [18]. The system were successful in estimating the COG and position of fingertips, however, did not work in real time.

Freeman et al. in 1995 presented a method based on a pattern recognition technique employing histograms of local orientation [19]. They used the orientation histogram as a feature vector for gesture classification and interpolation. The main contribution of this method is the invariance to different environment illuminations (See Figure 1.3). Later, this technique was patented under a static and dynamic gesture recognition system [20].

Because of the limited computing power, dynamic gestures were difficult to analyze. However, steady development of computer processing power along with better camera technology incrementally moved research towards dynamic

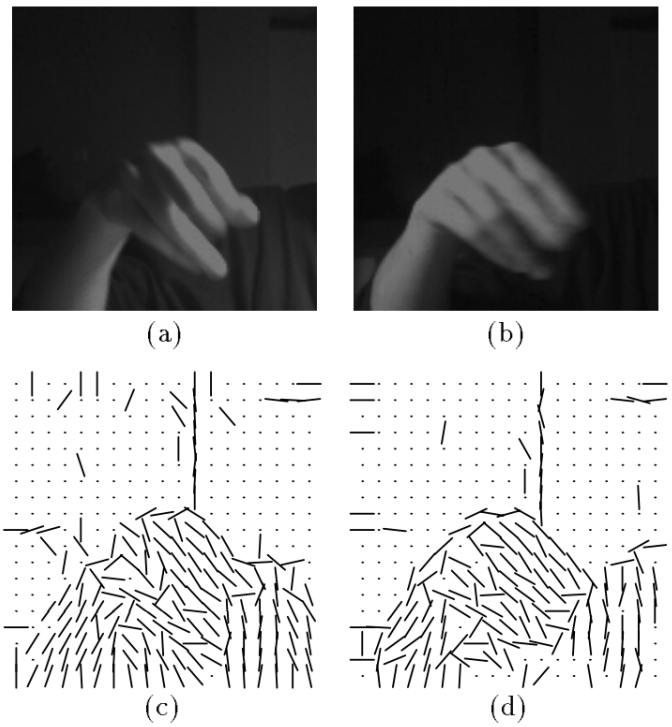


Figure 1.3: Showing the robustness of local orientation to lighting changes. Pixel intensities are sensitive to lighting change. (a) and (b) show the same hand gesture illuminated under two different lighting conditions. The pixel intensities change significantly as the lighting changes. Maps of local orientation, (c) and (d), are more stable. (Figure and caption reproduced from [19])

gesture recognition. One of the obstacles in dynamic gesture recognition was to identify information from a video which contained moving gestures. In 1995, Cui et al. demonstrated that this could be attempted by automatic visual partitioning [21]. They theorized that movement of the hand gesture could be decomposed into two components: global and local motions. The global motion captured gross motion in terms of position and the local motion characterized deformation, orientation and gesture changes.

Some of the many drawbacks using a single camera system include occlusion and the difficulty in removing the cluttered background [4]. Many researchers around 1998 started using multiple cameras to mitigate the effect of these drawbacks.

Shimada et al. proposed a method to estimate the pose (joint angles) of a mov-

ing human hand and check whether that was a valid gesture using 3D model [22]. Their approach offered an alternative to remove the occlusion problem using a single camera without using any depth information. In their approach, given an initial rough shaped 3-D model, possible pose candidates are generated in a search space efficiently reduced using silhouette features and motion prediction. Then, they selected the candidates with high posterior probabilities to obtain the rough poses. The feature correspondence was established even under quick motion and self-occlusion.

The solution of occlusion problem is the use of multiple cameras that will increase the field of view over the gesture, impossible to be recognized with a single camera. Nevertheless, a calibration process and other constraints are necessary in order to determine that multiple views belong to the same gesture at a given time. When two cameras are used, stereo vision strategy easily solves the correspondence problem. As we notice, the requirements and restrictions imposed by hand morphology and environment make that cameras evolve into what we nowadays known as RGB-D low-cost sensors (See section 1.3.5).

In 1998, Segen and Kumar of Bell Laboratories developed a novel multi dimensional hand gesture interface for HCI [23]. Their system consisted of two cameras (stereo setup) able to compute five spatial parameters (position and orientation in 3D) for index finger and thumb and track the hand at a frame rate of 60 Hz. The system responded to four types of gestures: *Point, Reach, Click* and *Ground*. A similar work was carried out by Utsumi and Ohya in 1999 [24]. Utsumi proposed a method of tracking 3D position, posture, and shapes of human hands from multiple-viewpoint images. A viewpoint selection mechanism was used in order to reduce self-occlusions and hand-hand occlusions. In order to track each hand position, Utsumi used Kalman filters. The small number of extracted features from images were based on distance transformation and they were robust against changes in hand shape and self-occlusions.

In 2003, Chen et al. [25] introduced a hand gesture recognition system to recognize continuous gestures (dynamic gestures). The system consisted of a real time hand tracking and extraction, a feature extraction module, hidden Markov

model (HMM) training, and gesture recognition. First, they applied a real-time hand tracking and extraction algorithm to trace the moving hand and extract the hand region, then they used the Fourier descriptor (FD) to characterize spatial features and the motion analysis for the temporal features. At the same time, they combine the spatial and temporal features of the input image sequence as the feature vector. After having extracted the feature vectors, they used HMM to recognize the input gesture. The gesture to be recognized is separately scored against different HMM. The model with the highest score indicates the corresponding gesture. In the experiments, they have tested the system to recognize 20 different gestures, and the recognizing rate is above 90%.

1.3.4 Hand gesture recognition system restrictions

In this subsection we will discuss the restrictions of a gesture recognition system regarding to hand morphology and the environment where HCI system would be expected to operate [1]. As we known, the hand is an **articulated body part** with more than 20 DOFs. We can move independently each finger, so we can perform a large variety of movements and hand poses. Due to, we will need to estimate a large number of parameters in order to calculate the orientation and location of each finger and hand itself. This fact clearly difficult the real-time gesture recognition process and the creation of a robust system.

By performing a complicated hand posture or a complex movement, **self-occlusions** may occur (e.g. a finger is occluded by another one or occlusions between other hand parts). In this case the hand segmentation and the extraction of high level features is much more complicated. Feature vector will increase in size that is directly proportional to the computational complexity, hindering the real-time computing.

We must consider that the **processing speed** is very important. The goal is a system capable of operating in real-time. A huge amount of data is processed by each image sequence, so some applications are quite demanding in terms of computational power. In order to meet those requirements, many applications

run on specific hardware architectures, for the most part, parallel architectures.

For widespread use, many HCI systems would be expected to operate under non-restricted backgrounds (**uncontrolled environments**) and a wide range of lighting conditions. Even locating a rigid object in an arbitrary background is almost always a challenging issue in computer vision. Segmentation process in cluttered scenes is more complicated and sometimes inaccurate.

The hand has **very fast motion capabilities** with a speed reaching up to 5 m/s for translation and 300^o/s for wrist rotation. Currently, off-the-self cameras can support 30-60 Hz frame rates. Besides, it is quite difficult for many algorithms to achieve even a 30 Hz tracking speed. In fact, the combination of high speed hand motion and low sampling rates introduces extra difficulties for tracking algorithms (i.e, images at consecutive frames become more and more uncorrelated with increasing speed of hand motion).

Satisfy all the issues listed above simultaneously is a very difficult task and it is a real challenge for the research in gesture recognition for HCI. Some studies apply restrictions on the user or the environment, for example, assume that the background is uniform or static and the hand is the only skin-colored object. However, such restrictions may not be applicable in many real-life systems. Another common restriction comes from rapid hand motions. For example, [26] based on its practical experience with high DOF control and manipulation tasks, recommends at least 100Hz tracking speed to allow effective interaction. There are many hand movements and poses involving plane rotation that consequently produce occlusions between fingers. Therefore, a common restriction to solve this issue is to assure that the palm is parallel to the image plane in order to minimize occlusions and avoid plane rotation.

According to [8], the most important attributes of a new human-computer interaction system are:

- High accuracy
- Ease of use without holding or wearing any devices in hand
- Shorter user learning cycle
- Lower cost
- Offer capabilities that are not available with any traditional interface

1.3.5 Hand gesture recognition system with low-cost sensors

Low-cost RGB-D cameras as *Microsoft Kinect* and *PrimeSense* are novel sensing systems used to capture RGB images along with per-pixel depth information. Before the emergence of 3D sensors, the predominant system for 3D capture was the laser scanning (See Figure 1.4), characterized as a costly system both temporarily as economically. These drawbacks clearly indicate that traditional systems such as laser scanning are not a feasible option as technologies in gesture recognition systems under demanding time constraints.

Figure 1.4: Example of laser scanning system, *SICK S3000* laser scanner. (Figure reproduced from `www.sick.com`).

In contrast to the laser scanning, there were other similar systems to the RGB-D cameras, known as high-performance 3D sensors based on time-of-flight (ToF) methods. Example of these sensors are *SR4500* and *SR4000* from *Mesa Imaging* [27] which provide depth maps by using modulated light source (see Figure 1.5). ToF based cameras are applicable to systems with demanding time constraints, nevertheless, the main problem of these sensors are the scarce possibilities of acquisition because of its high economic cost and reduced availability as in the case of laser scanning systems.

Nowadays, cutting-edge technology for computer vision applications are the low cost 3D sensors. The advent of these new RGB-D cameras put aside the powerful laser scanners and also reduces the importance of ToF based sensors. In

Figure 1.5: *Mesa Imaging SR4000* on the left side and *Mesa Imaging SR4500* on the right side as an example of time-of-flight sensors (Both figures were reproduced from [27]).

Figure 1.6: Left: *PrimeSense Carmine* sensor (Figure reproduced from `primesense.com`). Right: *Asus Xtion PRO*. (Figure reproduced from `asus.com`)

contrast to laser scanners and ToF sensors characterized by their high economic costs and reduced availability, low cost 3D sensors solve these two major limitations. *Kinect* sensors are the best example of success of these devices. Its main advantages are its small size, low power consumption and easy acquisition for a rather reduced market price. The price-performance ratio is unbeatable. In the same situation is the PrimeSense sensor (See Figure 1.6). In fact, the per-pixel depth sensing technology that is used in consumer RGB-D cameras was developed by PrimeSense (United States Patent US7433024) [28].

At the same time, this technology is licensed for use in the commercially-available *Microsoft Kinect* and *Asus Xtion PRO* (See Figure 1.6) sensors, both created as consumer products for natural user interface (NUI) applications. In particular, the *Kinect* is an Xbox accessory and was released in November, 2010. Cur-

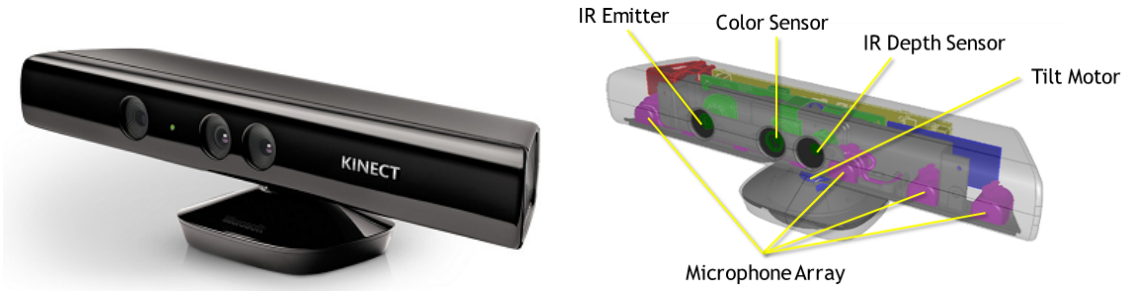


Figure 1.7: First version of *Kinect* sensor. On the right side we can see its main components. Both figures reproduced from `https://msdn.microsoft.com/en-us/library/hh855355.aspx`

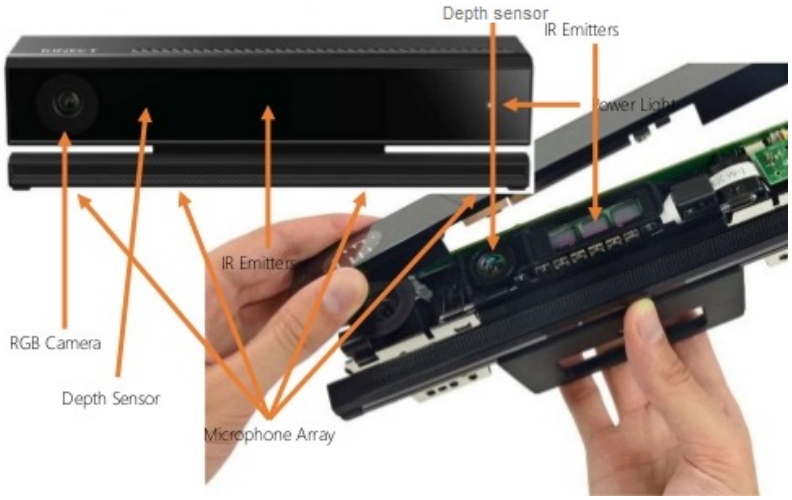


Figure 1.8: Second version of *Kinect* sensor with its main components. Figure reproduced from `https://msdn.microsoft.com/en-us/library/hh855355.aspx`

rently two versions of the *Kinect* sensor are available (*Kinect v1* 1.7 and *Kinect v2* (See figure 1.8)), and both can be used with computers.

1.4 Proposal

After describing the motivation of this work and analyzing the state of the art of HCI systems that have been proposed, we propose the implementation of a human-computer interaction system which will operate over skeleton data streams provided by Kinect v2, as a low-cost 3D sensor. We will use the Gesture Recognition Toolkit (GRT) described in section 2.2.1.1 in order create our gesture recognition pipeline and finally to train and test our complete gesture recognition system. The system will be optimized in order to achieve real-time results.

1.5 Goals

The main goal of this project is the development of a gesture recognition system for human-computer interaction. First of all, an extensive state of the art of gesture recognition systems will be carried out. Then we will analyze the Gesture Recognition Toolkit which will be our main tool for the development of this project. We will also describe all the Schaeffer gestures that our system will recognize asking for help and performing several visits to the APSA association of Alicante. When the evaluation of the gestures is finished, we proceed to the define our gesture recognition pipeline and at the same time analyze the process of data acquisition using Microsoft Kinect sensor. We also need to define our feature vector processing the raw data that Kinect provides us. The next goal will be related to choose and describe the machine learning algorithm that we will use in the training and classification processes. Finally we proceed to evaluate the performance of our system and perform a cross validation technique for the experimentation.

1.6 Structure

This document is structured as follows: Chapter 1 outlined the project and presented a detailed state of the art of each issue covered in this work. It exposed the motivation of the work and its general and specific goals. Chapter 2 presents the methodology followed during the development of this project. It exposes the different techniques, technologies and tools used to carry out this work. Chapter 3 presents our proposed solution. It first describes the proposal in depth. Then it presents the main implementation of our system. The experiments and discussion for each part of the implementation are detailed throughout the chapter. Finally, Chapter 4 details the conclusions extracted from this project and draws future work and research directions.

Chapter 2

Methodology

In this chapter we will describe the methodology employed in this work. It is organized into three different sections. Section 2.1 introduces the purpose of this chapter. Section 2.2 explains the multiple technologies that were used to obtain data, develop our applications or even improve them. At last, Section 2.3 describes the experimentation methodology that was followed to assess the different gesture recognition systems or phases.

2.1 Introduction

In order to carry out this project we rely on different technologies both software and hardware. Section 2.2 describes both kinds of technologies that were used during the development of the project. In order to compare and test the different approaches of this project we will specify a concrete methodology used for the experimentation 2.3.

First we will need a data acquisition system to feed our applications with skeleton data streams, color and depth information. We will also need software tools for developing those applications. We will use a gesture recognition toolkit that provides us different machine learning techniques with pre-processing and post-processing data methods and at the same time different tools in order to plot the results.

2.2 Technologies

In this section we will describe the different technologies which were directly or indirectly used during the development of this project. Subsection 2.2.1 describes the software tools and Subsection 2.2.2 focuses on hardware platforms and devices.

2.2.1 Software

The gesture recognition system will be implemented using C/C++ and C# as main programming languages. Our implementation was performed using Visual Studio 2013 as IDE and integrated with QT5 as add-in. We make use of QT5 in order to design the user interface. CMake will be used for managing the build process of the project and for compiling the *Gesture Recognition Toolkit* (GRT) or the *Open Sound Control* (OSC) library used for sending the data stream to our application. In order to manage the acquisition devices like *Microsoft Kinect*, we will make use of Kinect for Windows SDK 2.0 to obtain the necessary data flow. Since the SDK is only compatible with *Microsoft Windows* systems, our system has not been tested under other platforms.

The *Gesture Recognition Toolkit* (GRT) will be used to process the information provided by the sensors. This toolkit will provide us with different preprocessing, feature extraction and classification modules in order to implement our gesture recognition pipeline. The toolkit is designed to greatly simplify the process of building a new gesture-recognition system, with a focus on a low entry level for novice machine-learning users, while still remaining highly flexible for expert users. At the same time, we can easily extend it with our own algorithms. This toolkit will provide us with all the functionality and necessary tools to implement our gesture recognition system at a higher level. Since the purpose of this work is not to implement the used algorithms, we will not reinvent the wheel, reimplementing such complex modules.

2.2.1.1 Gesture Recognition Toolkit

The *Gesture Recognition Toolkit* (GRT) is a cross-platform open-source C++ library available under MIT license and designed to make real-time machine learning and gesture recognition more accessible for non-specialists [29]. As it is already known, gesture recognition is a powerful tool for human-computer interaction (HCI), and it is characterized by its increasing prevalence in principal consumer devices as smartphones, wearable devices, televisions and gaming consoles. The fact of its growing appearance on current technologies justifies the need of a tool or API that helps non-specialists to build custom gesture-based applications. The toolkit features a broad range of classification, regression, time-series algorithms and has extensive support for building real-time systems. This includes algorithms for signal processing, feature extraction and automatic gestures spotting as we will see in this section.

Core design principles

The toolkit was developed with the following core design principles [29]:

Accessibility: The *GRT* is a general-purpose tool for facilitating non-specialists to create their own machine-learning based systems. Ease of use is one of its main objectives, along with a clear and consistent coding convention. This consistent, minimalist design significantly lowers the entry barrier for a new user.

Flexibility: The *GRT* uses an object-oriented modular architecture in order to support flexibility while maintaining consistency. This architecture is built around a set of core modules and a central gesture-recognition pipeline. This toolkit includes module for preprocessing, feature extraction, clustering, classification, regression and post processing. The general input to both core modules and pipeline consists of an *N-dimensional double-precision vector*, making the tool flexible to the type of input signal.

Choice: There is no single machine-learning algorithm that can be used to recognize all gestures. The *GRT* features a broad range of machine-learning algorithms such as AdaBoost, Decision Trees, Dynamic Time Warping, Hidden Markow Models, K-Nearest Neighbor, Linear and Logistic Regression, Naïve

Bayes, Multilayer Perceptrons, Random Forests, Support Vector Machines and more. This variety of algorithms greatly facilitates the experimentation and comparison processes, between the algorithms chosen by the user. Switching between the machine-learning algorithms is a trivial and easy process, and not involves significant modifications by the user.

Supporting Infrastructure: There are other important things besides the machine learning algorithms, such as preprocessing and feature extraction. The *GRT* supports a wide range of pre/post processing, feature extraction and feature selection algorithms, including popular preprocessing filters (e.g., Moving Average Filter), embedded feature extraction algorithms (e.g., AdaBoost), dimensionally reduction techniques (e.g., Principal Component Analysis), and unsupervised quantizers (e.g., K-Means Quantizer, Self-Organizing Map Quantizer). The toolkit also contains support for recording, labelling and managing supervised and unsupervised data sets for classification, regression and time series analysis.

Customizability: The *GRT* is designed to allow users to easily incorporate their own algorithms within the toolkit framework by inheriting from one of the GRT base classes. In this way, more advanced users can easily include and test their own implementations, such as custom feature-extraction algorithms.

Real-time Support: The *GRT* supports common techniques for performing offline analysis on pre-recorded data sets, such as partitioning data into validation and test data sets, running cross validation and computing accuracy metrics. Besides, the toolkit is designed to enable a user to seamlessly move from offline analysis phase to the real-lime recognition phase. To support real-time gesture recognition, the GRT features algorithms that automatically perform gesture spotting. These algorithms, such as the Adaptive Naïve Bayes Classifier and N-Dimensional Dynamic Time Warping, learn rejection thresholds from the training data, which are then used to automatically recognize valid gestures from a continuous stream of real-time data.

The toolkit consists of two parts: a comprehensive C++ API and a front-end graphical user interface (GUI) as shown in Figure 2.1. The input to the GRT can be any N-dimensional floating-point vector. This means you can use the toolkit

with Cameras, Kinect, Leap Motion, accelerometers, or any other custom sensor you might have built.

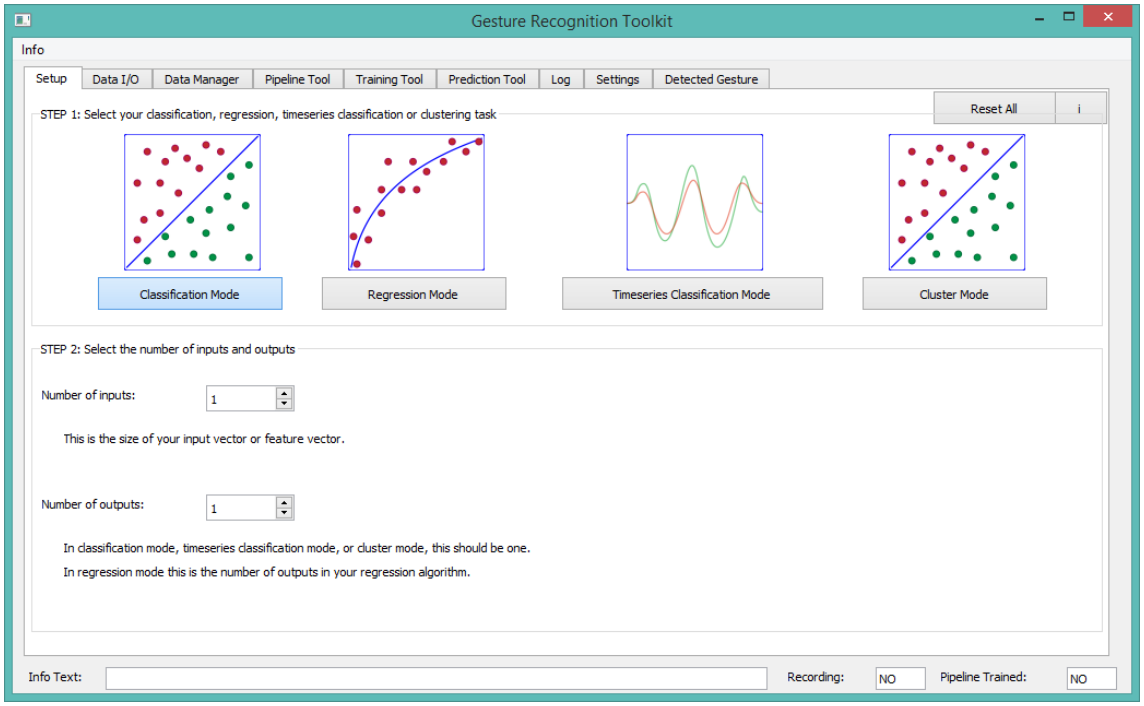


Figure 2.1: GRT graphical user interface. The input data to the GUI is streamed via Open Sound Control (OSC) from an auxiliar application. The entire gesture recognition pipeline is easily configurable using this user interface. The GUI is implemented on QT (Figure reproduced from [30]).

The GRT has been designed to [30]:

- be easy to use and integrate into your existing C++ projects
- be compatible with any type of sensor or data input
- be easy to rapidly train with your own gestures
- be easy to extend and adapt with your own custom processing or feature extraction algorithms (if needed)

The GRT features a large number of algorithms that can be used to [30]:

- recognize static postures (such as if a user has their hands in a specific posture or if a device fitted with an accelerometer is being held in a distinct orientation)
- recognize dynamic temporal gestures (such as swipe or tap gesture)

- perform regression (i.e. continually map an input signal to an output signal, such as mapping the angle of a user's hands to the angle a steering wheel should be turned in a driving game)

The current development build of the GRT contains machine-learning algorithms such as:

1. Adaptative Naive Bayes Classifier
2. AdaBoost
3. Decision Tree
4. Dynamic Time Warping
5. Gaussian Mixture Models (GMM)
6. Hidden Markov Model
7. K-Nearest Neighbor Classifier
8. MinDist
9. Random Forests
10. Softmax
11. Support Vector Machine
12. Artificial Neural Network
13. Linear Regression
14. Logistic Regression
15. Multidimensional Regression

The GRT also contains a large number of pre-processing, post-processing, and feature-extraction algorithms such as:

1. Low Pass Filter
2. High Pass Filter
3. Moving Average Filter
4. Double Moving Average Filter
5. Derivative
6. Dead Zone
7. Savitzky-Golay Filter
8. Zero Crossing Counter
9. FFT
10. FFT Features
11. KMeansQuantizer
12. Movement Trajectory Features
13. Principal Component Analysis
14. Class Label Filter
15. Class Label Change Filter
16. Class Label Timeout Filter

Gesture recognition pipeline

There are generally six main steps for creating a gesture recognition system using the GRT:

1. Select suitable algorithms to solve the gesture recognition problem
2. Setting up a gesture recognition pipeline
3. Recording the training data
4. Training the pipeline
5. Testing the recognition accuracy of the pipeline
6. Use the pipeline for the real-time prediction

1. Select suitable algorithms to solve the gesture recognition problem The first step consists in selecting the most suitable pre-processing, feature-extraction, recognition, and post-processing algorithms that might work well to solve your gesture recognition problem. One of the most common stumbling block for non-experimented users who want to create a gesture recognition system is the selection of the most suitable machine-learning algorithm. The main indicator of this election is the output of our system. If the output consists of a specific value (i.e. someone else has just made a gesture) we are trying to solve a classification problem. Alternatively, if it consist of a continuous value or values, we are trying to solve a regression problem. An example of a classification problem might involve giving your system an input image, captured from a webcam for instance, and the job of the system is to classify a specific user's face from this within this image. Whereas an example of a regression problem might involve giving your system an input image, perhaps captured from a camera on top of a moving car, and the job of the system is to estimate what angle the car's steering wheel should be turned to keep the car on the road.

Both regression problems as classification are quite different. However, our system will face only with classification ones. We can break classification problems into two sub-categories, that of recognizing static postures and recognizing

temporal gestures. The difference between these two sub-categories is clearly reflected in the subsection 1.3.2.

The GRT has a number of algorithms that are suitable for both static posture and temporal gesture recognition, examples of these are:

- Static Posture Recognition Algorithms
    1. K-Nearest Neighbor Classifier (KNN): A very simple classifier that works well on basic recognition problems, can be slow for real-time prediction though and is not robust to noisy data.
    2. Adaptative Naive Bayes Classifier (ANBC): A naive but powerful classifier that works very well on both basic and more complex recognition problems.
    3. Support Vector Machine (SVM): A very powerful classifier that works very well on complex classification problems.
- Temporal Gesture Recognition
    1. Dynamic Time Warping (DTW): A powerful classifier for temporal gestures, not really suitable for gestures that are not temporal.

The KNN, ANBC, and SVM algorithms can also be used for dynamic gesture recognition if they are paired with a suitable feature-extraction algorithm that can take a temporal signal and compute some relevant features from this, which can then be input into the aforementioned classifiers.

2. Setting up a gesture recognition pipeline

After selecting the algorithms that might be suitable for your gesture recognition problem, the next step is to setup a new gesture recognition system which you can train and use to recognize your own gestures. GRT is basically a gesture recognition pipeline that serves as a container to which you can add the classification or regression algorithm that you think might work best for solving your gesture recognition problem. The input of the pipeline can be your own sensor data while the output will be the predicted class label of a gesture from the end of the pipeline. You also can add a number of pre-processing, feature-extraction, and

post-processing algorithms in order to make the classification results more accurate by mitigating false-positive classification errors or by stopping one gesture from being recognized a number of times. Each of this algorithms is a module of the pipeline which together result in a complete gesture recognition pipeline as we can see in the Figure 2.2.

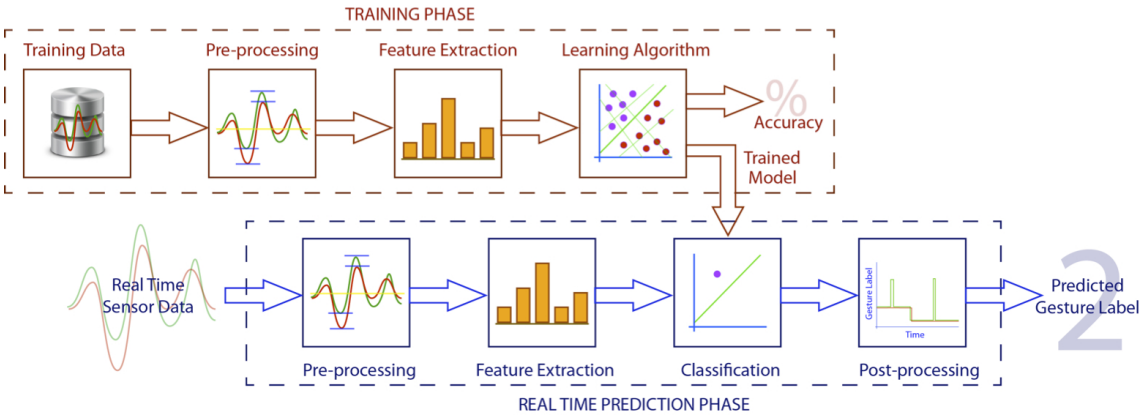


Figure 2.2: Modularity of the GRT gesture recognition pipeline. The connexion between the modules during the recognition process is represented (Figure reproduced from [30]).

Listing 2.1 demonstrates how to create a new gesture recognition pipeline, using an ANBC classification algorithm at the core of the pipeline, supported by a number of pre-processing, feature-extraction, and post-processing modules.

Listing 2.1: Creating a new gesture recognition pipeline example

```
1 //Create a new gesture recognition pipeline
2 GestureRecognitionPipeline pipeline;
3
4 //Moving average filter as a pre-processing module to the start
5 //of the pipeline (buffer size of 5 for a 1 dimensional signal)
6 pipeline.addPreProcessingModule( MovingAverageFilter(5,1) );
7
8 //FFT (window size of 512) as a feature-extraction module, the input
      to
9 //this module is the output of the moving average filter.
10 pipeline.addFeatureExtractionModule( FFT(512) );
11
12 //Add a custom feature module to the pipeline,
13 pipeline.addFeatureExtractionModule( MyOwnFeatureMethod() );
14
15 //Set the classifier at the core of the pipeline, in this case we
     are
```

```
16 //using an Adaptive Naive Bayes Classifier
17 pipeline.setClassifier( ANBC() );
18
19 //Add a class label timeout filter to the end of the pipeline (with
       a
20 //timeout value of 1 second), this will filter the predicted class
21 //output from the ANBC algorithm
22 pipeline.addPostProcessingModule( ClassLabelTimeoutFilter(1000) );
```

As we can see in the above example code, the creation and setup of a new gesture recognition pipeline is very intuitive and easy to understand. At the same time, its easy to integrate your own custom feature-extraction, pre-processing and post-processing modules into any pipeline (e.g. *MyOwnFeatureMethod()* custom method). This custom feature-extraction module might, for example, take the output of the FFT module as its input and compute some features from the FFT signal, such as the top N frequency values.

After setting up our custom gesture recognition pipeline we can train it, and finally use the trained pipeline for real-time gesture recognition.

3. Recording some training data

Before recognize real-time gestures we will need to train our gesture recognition pipeline. In order to train the pipeline we will need a training dataset. For that, we can easily record some examples of the gestures we want the pipeline to recognize. The GRT has a number of utilities and data structures useful for record, label, manage, save and load training data.

For example, in the case of ANBC, KNN, SVM or GMM classifier at the core of the pipeline, we will record the training data using the *ClassificationData* structure. We can also load CSV data directly from a file using the same data structure. This feature will allow us to edit, record and label our training data in another program (such as Excel or Matlab). Listing 2.2 demonstrates how to create a simple training data in few steps.

Listing 2.2: Creating a simple training data example

```
1  //Create a new instance of the ClassificationData
2  ClassificationData trainingData;
3
4  //Set the dimensionality of the data
5  trainingData.setNumDimensions( 3 );
6
7  //Here you would grab some data from your sensor and label it with
8  //the corresponding gesture it belongs to
9  unsigned int gestureLabel = 1;
10 vector< int > sample(3);
11 sample[0] = //....Data from sensor
12 sample[1] = //....Data from sensor
13 sample[2] = //....Data from sensor
14
15 //Add the sample to the training data
16 trainingData.addSample( gestureLabel, sample );
17
18 //Saving recorded training data to file with .grt extension
19 bool trainingDataSaved = trainingData.save(                    );
20
21 //Loading training data
22 bool trainingDataLoaded = trainingData.load(                    )
      ;...
```

There are more data structure available such as *TimeSeriesClassificationData* considered as the main data structure for recording, labeling, managing, saving and loading training data for supervised temporal learning problems. Unlike the *ClassificationData*, in which each sample consists of 1 N dimensional datum, a *TimeSeriesClassificationData* sample will consist of an N dimensional time series of length M. We will use this data structure to train our *Dynamic Time Warping* (DTW) algorithm.

4. Training the pipeline

After recording the training data we can easily train our classification algorithm at the core of the pipeline as in Listing 2.3. The code is the same for all machine-learning algorithms, either classification or regression.

Listing 2.3: Pipeline training example

```
1 //Train the pipeline
2 bool trainSuccess = pipeline.train( trainingData );
```

5. Testing the recognition accuracy of the pipeline

After training the pipeline we can test it in order to validate how well it will work with the new data as in Listing 2.4.

Listing 2.4: Pipeline test process example

```
1  //Test the classification accuracy of the trained pipeline
2  bool testSuccess = pipeline.test( testData );
3
4  //Get the accuracy of the pipeline performed with the test data
5  double accuracy = pipeline.getTestAccuracy();
6
7  //Some other results such as the F-Measure, Precision and Recall
8  double fMeasure = pipeline.getTestFMeasure();
9  double precision = pipeline.getTestPrecision();
10 double recall = pipeline.getTestRecall();
```

Alternatively, if you don't have enough training data to create a separate test dataset you can train the pipeline using k-fold cross validation and the existing training data as showed in Listing 2.5.

Listing 2.5: K-fold cross validation technique

```
1  //Perform the prediction with the trainingData and 10 folds
2  bool trainSuccess = pipeline.train( trainingData, 10 );
3
4  //You can then get then get the accuracy of how well the pipeline
5  //performed during the k-fold cross validation testing
6  double accuracy = pipeline.getCrossValidationAccuracy();
```

The k-fold cross validation example code does not work with DTW algorithm since this technique is not yet implemented for this machine learning algorithm.

We can also train our pipeline using random test subset or an external test dataset. Our system was trained using k-fold cross validation in order to ensure that the training partitions and test data are independent. The problem is that k-fold cross validation method doesn't work properly using the GUI of GRT. A series of bugs were reported to the developer.

6. Using the pipeline for real-time prediction

After training the pipeline the last step consists of performing real-time prediction in order to predict the class label for the new input data. Listing 2.6 shows

an implementation of this final step.

Listing 2.6: Real-time classification with new input data example

```
1  //Perform the prediction
2  bool predictionSuccess = pipeline.predict( inputVector );
3
4  //You can then get the predicted class label from the pipeline
5  UINT predictedClassLabel = pipeline.getPredictedClassLabel();
6
7  //Along with some other results such as the likelihood of the most
8  //likely class or the likelihood of all the classes in the model
9  double bestLoglikelihood = pipeline.getMaximumLikelihood();
10 vector<double> classLikelihoods = pipeline.getClassLikelihoods();
11
12 //You can then use the predicted class label to trigger the action
13 //associated with that gesture
14 if( predictedClassLabel == 1 ){
15 //Trigger the action associated with gesture 1
16 }
17 if( predictedClassLabel == 2 ){
18 //Trigger the action associated with gesture 2
19 }
```

2.2.1.2 Kinect for Windows SDK 2.0

The *Kinect for Windows SDK 2.0* is a software development kit from *Microsoft* that contain header files, libraries, samples, documentation and tools required to develop applications for *Microsoft Windows* using the Kinect v2 sensor as input device. The SDK is available for free on the *Microsoft Download Center*, installs quickly and requires no complex configuration. Developers can get up and running in just a few minutes with a standard standalone Kinect v2 sensor unit. The SDK will permit us the access to raw data streams from depth sensor, color camera sensor, and four-element microphone array. One of the most important features is called *Skeletal Tracking*. This functionality provides the capability to track the skeleton image of one or two people moving within the Kinect field of view make it easy to create gesture-driven applications. At the same time, the SDK includes more than 100 pages of technical documentation and different code samples ready to be used. This software was designed for non-commercial purposes.

2.2.1.3 SharpOSC - OSC Library for .NET 3.5

Open Sound Control (OSC) is a protocol for networking sound synthesizers, computers, and other multimedia devices. There are different implementations of OSC libraries. The main reason we will use this library is because, the GRT GUI features an OSC backend that enables other applications to send and receive data over a network. This enables other applications (like Max for instance) that might be running on either the same computer, or on another computer on the same network, to control the GRT GUI.

In our project we used a small library called SharpOSC implemented in C#. This library provides the following features:

- Produce and OSC Packet (messages and bundles) from .NET values
- Translate an OSC message (consisting of a sequence of bytes) into a .NET object
- Transmit OSC packets via UDP
- Receive OSC packets via UDP

The full implementation of this library can be downloaded from the GitHub repository (`https://github.com/ValdemarOrn/SharpOSC`). To use the library we will need to add a reference to SharpOSC.dll in our .NET project and then use SharpOSC namespace.

Listing 2.7 shows how to send an OSC message to the local machine (127.0.0.1) on port 50000 containing 3 arguments: an integer with a value of 23, a floating point number with the value 42.01 and the string "hello world". If another program is listening to port 50000 it will receive the message and be able to use the data.

Listing 2.7: Sending an OSC message example

```
1 var message = new SharpOSC.OscMessage(       , 23, 42.01f,
          );
2 var sender = new SharpOSC.UDPSender(          , 50000);
3 sender.Send(message);
```

2.2.2 Hardware

2.2.2.1 Microsoft Kinect

At present there are two *Microsoft Kinect* low-cost 3D sensors available in the market: *Kinect v1* (See Figure 1.7) and *Kinect v2* (See Figure 1.8). In this subsection we will analyze both sensors briefly, highlighting the similarities and differences between them. At the same time we will discuss the importance of these sensors in the context of hand posture and gesture recognition and we will also provide information about its performance and a good basis for understanding how these sensors work.

With the advent of the new low-cost 3D sensors, *Microsoft Kinect* has become one of the most popular sensors of the moment due to its high distribution and easy acquisition. It has received a lot of attention thanks to the rapid human pose recognition system developed on top of 3D measurement [31]. Its low cost, reliability and speed of measurement promise to make Kinect the primary 3D measuring device in indoor robotics, as in the case of *TurtleBot* robot [32], 3D scene reconstruction [33] [34] [35], object recognition [36] and gesture recognition [37], among others.

Microsoft Kinect software technology was developed internally by Rare, a subsidiary of *Microsoft Game Studios*, on a range camera technology engineered by Israeli developer *PrimeSense* [38].

Kinect features an infrared laser emitter, an infrared camera, an RGB one and a multi-array microphone (see sensor structure at Figure 1.7 and Figure 1.8). The laser emits a single beam which is split into multiples ones by a diffraction grating in order to create a constant speckle pattern. This pattern is projected onto the scene and its reflection is captured by the infrared camera. This principle is called **structured light** and basically consists on projecting a known pattern in form of light beams and infer depth from the deformation of that pattern using the light beams reflected back to the IR depth sensor. This technology was developed by *Primesense*. Figure 2.3 shows the speckles pattern projected by a Kinect device and the corresponding depth map for that image.

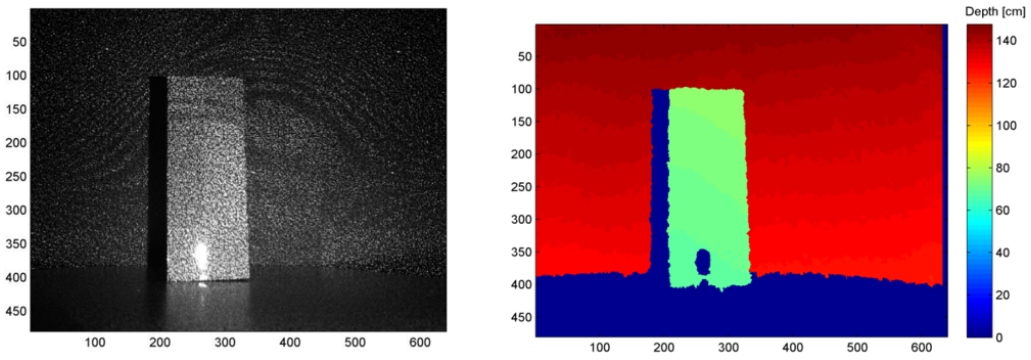


Figure 2.3: Left side: Infrared image of the pattern of speckles projected on a sample scene. Right side: The resulting depth map.

The depth measurement is performed by a process named depth triangulation [39]. First, a set of reference patterns are obtained by projecting those patterns on a plane at a know distance. Kinect and PrimeSense sensors use three patterns for three different regions of distance: 0.8 - 1.2 meters, 1.2 - 2.0 meters and 2.0 - 3.5 meters. The nearest region has a higher accuracy than the farthest one.

Later, the IR emitter projects those patterns onto unknown scenes. The speckles are then projected onto surfaces whose distance to the sensor is different than the one of the reference plane for that pattern. Because of that, the speckles on the captured infrared image will be shifted in the direction of the baseline between the IR emitter and the perspective center of the IR camera [40]. The shifts can be measured for all speckles using an image correlation procedure to generate a disparity image which can be used to compute the distance of each pixel to the sensor.

Figure 2.4 shows the mathematical model explained by Khoshelham et al. [40] which illustrates the relationship that exists between the distance of an object point k to the sensor and the measured disparity d, assuming a depth coordinate system in which the origin is the perspective center of the IR camera, the Z axis is orthogonal to the image plane, the X axis is perpendicular to the Z one in the direction of the baseline b and the Y axis is orthogonal to both of them creating a right handed coordinate system. As we have previously noted, if an object is displaced closer or further away from the sensor, the speckle is displaced in the

X axis direction thus allowing the measurement of the disparity.

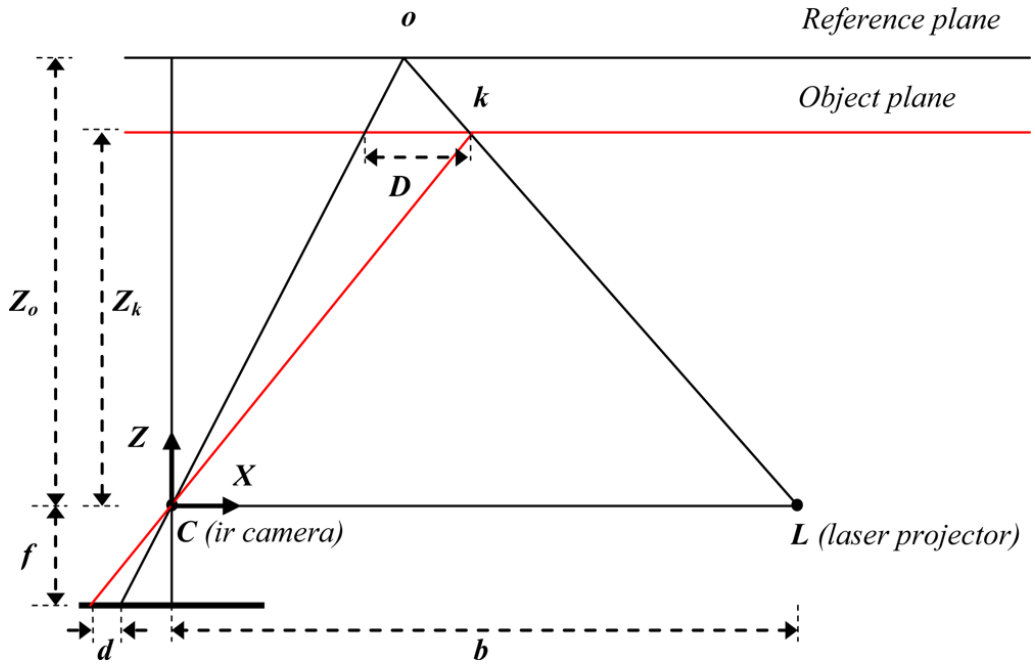


Figure 2.4: Relation between relative depth and measured disparity [40]

This system can provide full-body 3D motion capture, facial recognition and voice recognition capabilities. Body parts are inferred using a randomized decision forest, learned over 1 million training examples using a distributed implementation on a 1000 core cluster during about a day [31].

With these sensors we can also obtain a representation of the association between depth image and color image pixels. This association is a collection of points in three dimensional space or also called **point cloud**, where each pixel can have additional features associated with it, as color information or approximated surface normals.

Both Kinect sensors are quite different. Table 2.1 lists the main features of both Kinect versions.

As we can see, an improvement of the Kinect v2 sensor regarding tho the previous version is the higher resolution of the image and depth streams. Accordingly, horizontal and vertical fields of view have increased. At the same time, skeleton now is composed by more joints, including hand joints. This new camera makes use of the time of flight method. In this case, a clock signal strobes an

Features	Microsoft Kinect v1	Microsoft Kinect v2
Color Camera	VGA 640 x 480 @30 fps	1920 x 1080 @30 fps
Depth Camera	QVGA 320 x 240	512 x 424
Infra-Red (Spreads Dots)	YES	NO
Infra-Red (TOF)	NO	YES
Infra-Red Spectrum	827-850 nm	827-850 nm
Max Depth Distance	4.5 M	4.5 M
Min Depth Distance	40 cm in near mode	50 cm
Horizontal Field of View	57 degrees	70 degrees
Vertical Field of View	43 degrees	60 degrees
Tilt Motor	YES	NO
Vertical Tilt range	$\pm 27^{o}$	-
Skeleton Joints Defined	20 joints	26 joints
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0
Power Consumption	12 W	12 W
Supported OS	Windows 7 and 8	Windows 8

Table 2.1: Comparison of Microsoft Kinect devices.

array of laser diodes whose beams go through diffusers projecting short pulses of infrared light onto the scene. These pulses reflect on the different surfaces and are captured by the IR camera of the Kinect v2.

This IR camera is special because each pixel is divided into two parts which act as accumulators of photons of laser light. The aforementioned clock signal controls which half of each pixel is turned on and registering light pulses. Each half is 180 degrees out of phase from the other one, so that when the first is on the other is off and vice versa. At the same time, the laser light source is pulsed in phase with the first pixel half. In this sense, the clock signal determines which half is actively registering reflected light as shown in Figure 2.5. This method is named indirect Time-of-Flight and the distance can be inferred by comparing

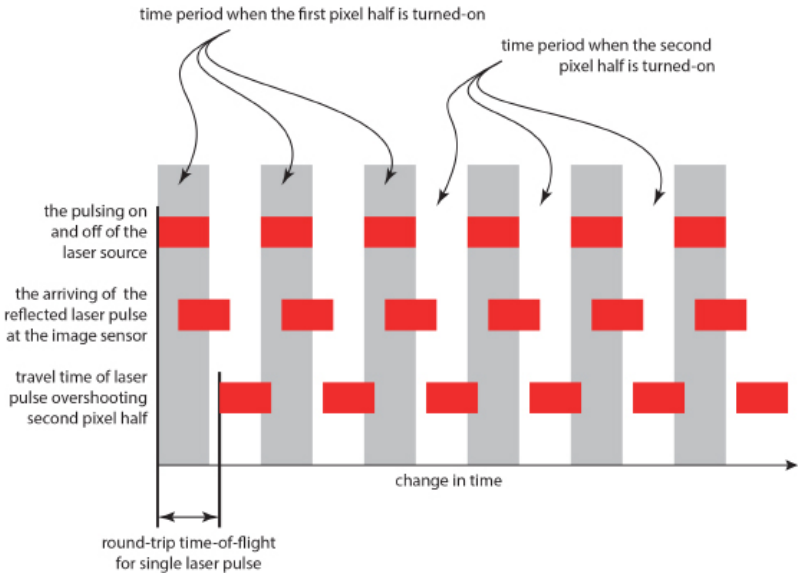


Figure 2.5: Illustration of the indirect Time-of-Flight method [41].

the ratio of light received in each pixel half, i.e., the more light that arrives in the second accumulator against the first, the farther that part of the scene is from the sensor [42].

2.3 Experimentation

In this section, we will briefly introduce the experimentation methodology which will be followed during the rest of this document.

2.3.1 Measuring performance

We will measure the performance of our gesture recognition pipeline regarding the accuracy obtained from the k-fold cross validation technique. We will analyze different pipeline configurations, using pre-processing and post-processing modules in order to observe the variation of the accuracy value. We will use 5-fold cross validation, and we will reproduce by each gesture and each fold the values for the f-measure, precision and recall.

We will also test the system diverse gesture datasets including different number of samples by each gesture. The experimentation results will be plotted in graphs or tables.

2.3.2 K-fold cross validation

In order to check that the obtained results are suitable in terms of accuracy, precision and coverage we used k-fold cross validation technique whose operation is illustrated in Figure 2.6.

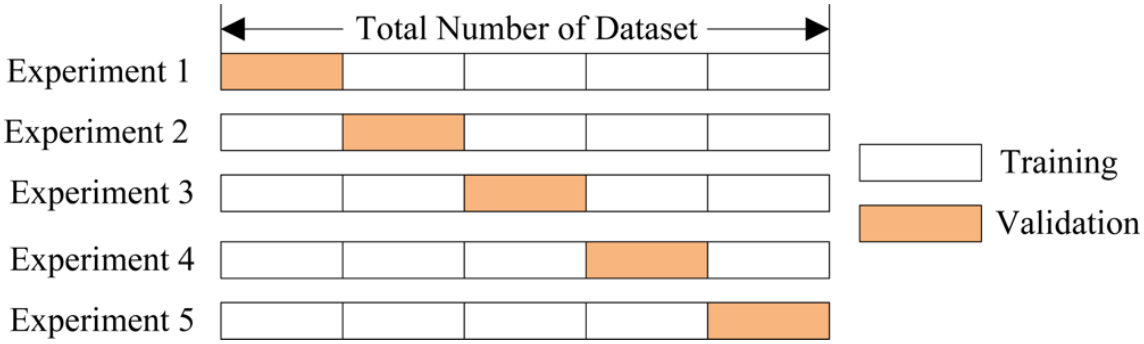


Figure 2.6: 5-fold cross validation, image reproduced from *Crop Classification by Forward Neural Network with Adaptive Chaotic Particle Swarm Optimization* [43].

As we can see, this validation method partitions the training set in k folds

and performs experiments in which a different set of test is selected each time, remaining the rest of folds as the training set. Finally. the results of each fold are averaged to obtain a measure of the algorithm. In this sense, a more formal definition of cross-validation is the following: it is a model validation technique to determine its generalizability in independent datasets. In short, it allows us to estimate how accurate will be a model in practice limiting the occurrence of problems such as overfitting by partitioning the data into training and test sets.

2.3.3 Test system

The test machine used for running the CPU related experiments is an Intel(R) Core i5-3570 CPU @ 3.40 GHz (4 CPUs), 3.4GHz with 8 GiB RAM DDR3 1866 MHz and an ASUS P8H77-M Pro motherboard with a chipset Intel H77. The operating system is Windows 8.1 Pro 64-bit (6.3, Build 9600). The system were compiled using Visual Studio 2013 Professional using Release settings for maximum speed optimization.

Chapter 3

Gesture recognition system

In this chapter we will describe the implementation of our system proposal introduced in Chapter 1. This chapter is organized in four sections. Section 3.1 introduces our proposed solution and a brief summary of the main content of this chapter. Section 3.3 shows the gesture recognition pipeline implementation. Section 3.2 is devoted to an in depth analysis of all the recognized gestures regarding their pros, cons and similarities. Finally, section 3.4 shows the experimentation which was carried out to validate our system and assess its performance.

3.1 Introduction

After reviewing the state of the art of different gesture recognition systems and defining the stages of a gesture recognition pipeline we proceed to perform our own implementation. The implementation of our gesture recognition pipeline is based on the *Gesture Recognition Toolkit* (GRT). The system will be programmed in C++ and C#, making use of different libraries such as *Open Sound Control* (OSC). An analysis of each gesture indicating their pros, cons and similarities will be performed. At the same time, the raw data from the Kinect will be analyzed in order to use only the relevant information for our recognition problem.

Moreover, a definition of the feature vector used as input will be reflected. Each feature has its own justification and is based on a concrete gesture. Finally a

study of our system performance was carried out in order to validate the system and determine its reliability.

3.2 Implemented Schaeffer gestures

In this section we will introduce a subset of Schaeffer gestures that our system will be able to recognize. The Schaeffer sign language is a simple language destined for disabled people in order to express themselves [6]. We will recognize 11 different gestures discussed below in detail. For each gesture we will include three timeseries graphs that represent three recorded samples of that concrete gesture. A sample is the execution of a gesture made by the user. The timeseries graphs represent the evolution in time of the values of each feature. For this, we will have one colored line for each component of feature vector.

As we will be able to observe, the three graphs are similar. That is because, the executions of the same gesture several times by the same user, are similar to each other. Nevertheless, there are small variations that are clearly visible when analyzing the timeseries plot. In order to compare gestures we can analyze the variation of the feature values. For a good classification, every individual gesture must have a significant variation in more than one feature. This variation is represented on the timeseries graph. In Section 3.3 we will compare the similar gestures and analyze them in order to achieve distinguish them on the prediction process.

3.2.1 Gesture 1: Help

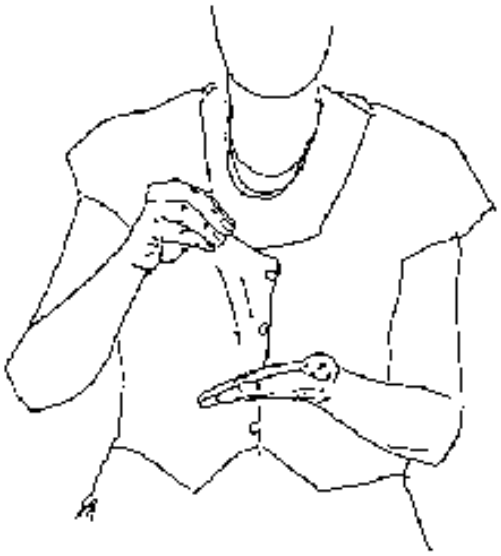


Figure 3.1: Help gesture: The fingertips of one hand together hitting the palm of the other hand.

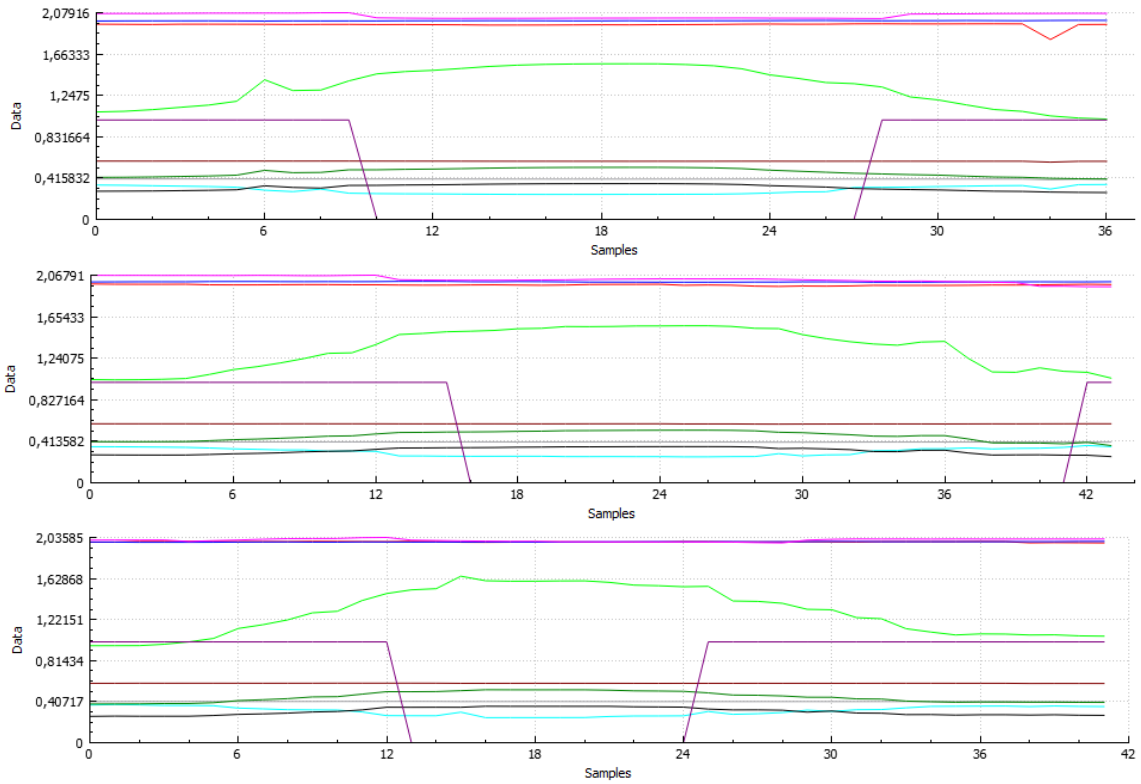


Figure 3.2: Gesture 1 timeseries plot of three samples.

3.2.2 Gesture 2: Water

Figure 3.3: Water gesture: Outstretched hand, with the thumb directed to the mouth. Make two movements towards this.

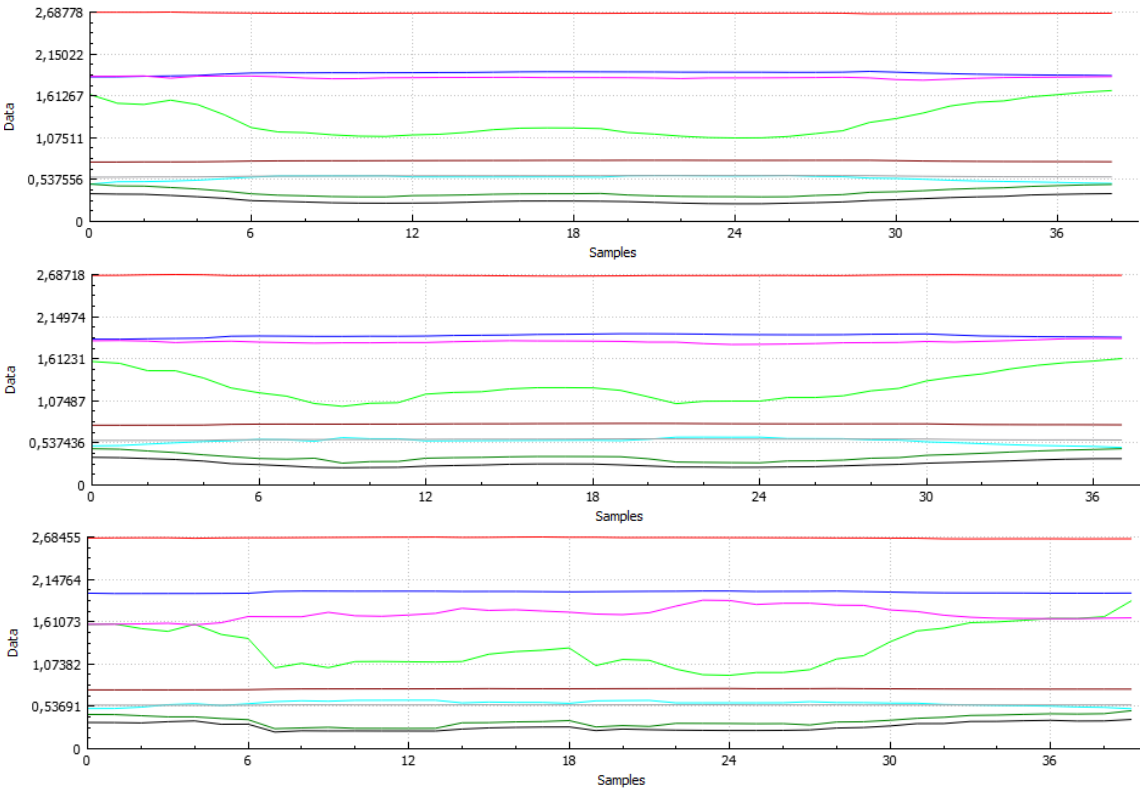


Figure 3.4: Gesture 2 timeseries plot of three samples.

3.2.3 Gesture 3: Snack

Figure 3.5: Snack gesture: Opened hand with inward palm comes near and move away the chin.

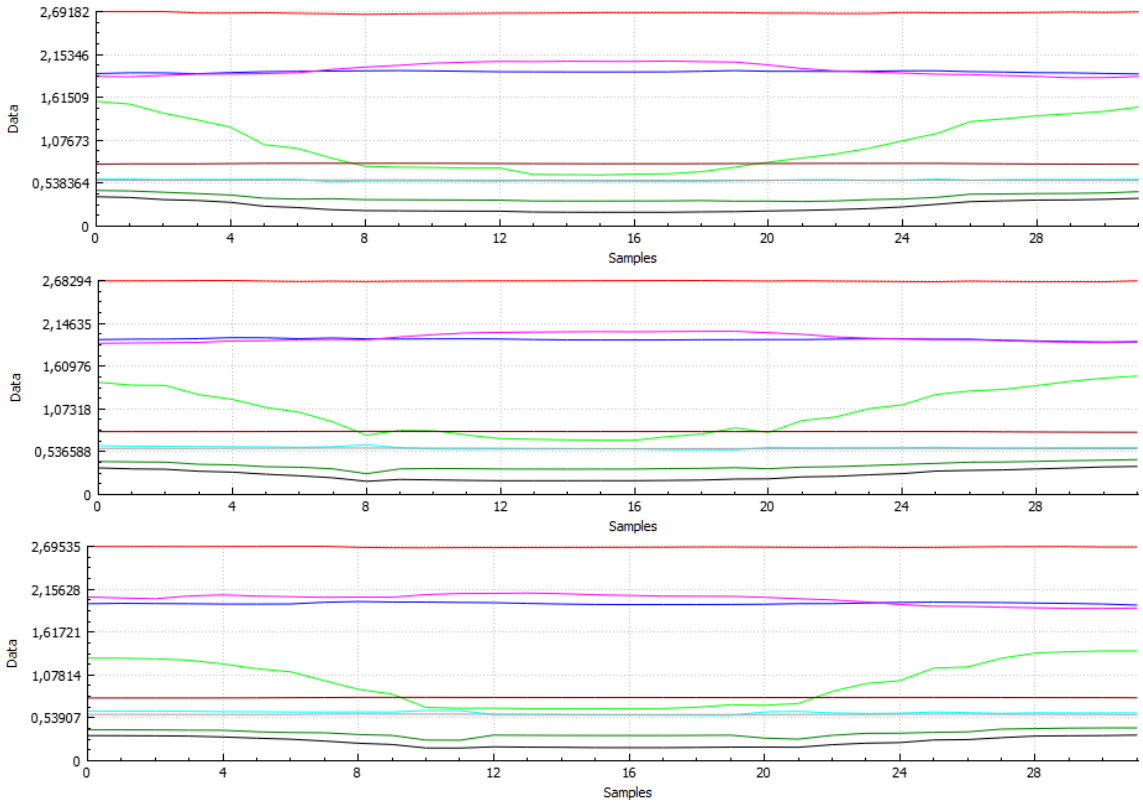


Figure 3.6: Gesture 3 timeseries plot of three samples.

3.2.4 Gesture 4: Sleep

Figure 3.7: Sleep gesture: Hands together live up to the face, head leans towards the hands.

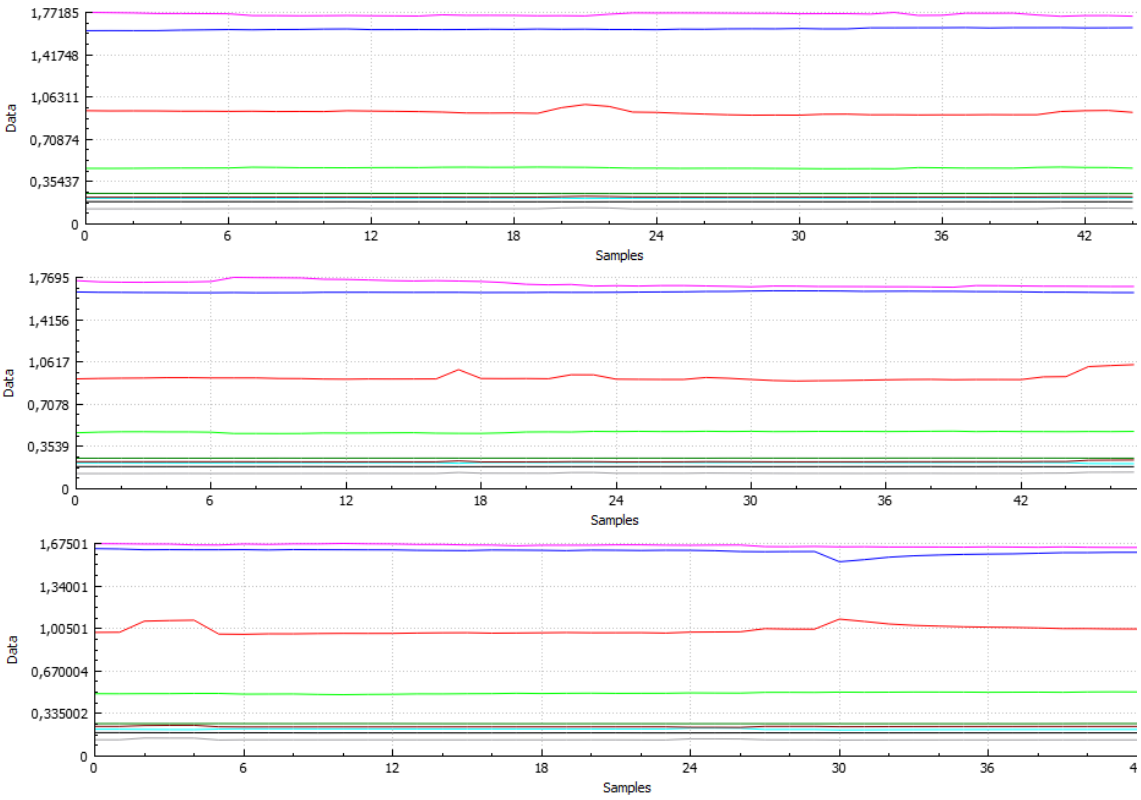


Figure 3.8: Gesture 4 timeseries plot of three samples.

3.2.5 Gesture 5: Showering

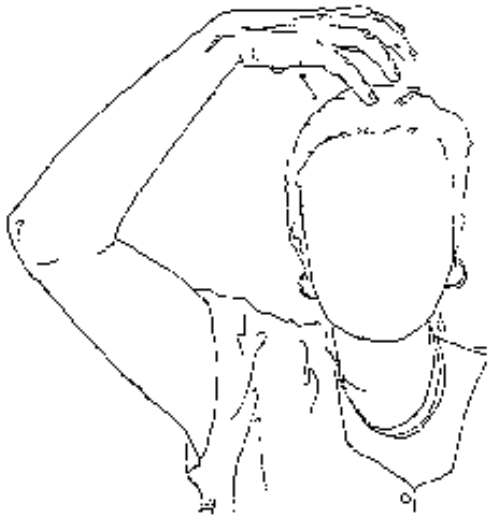


Figure 3.9: Showering gesture: Opened hand with palm above the head goes up and down.

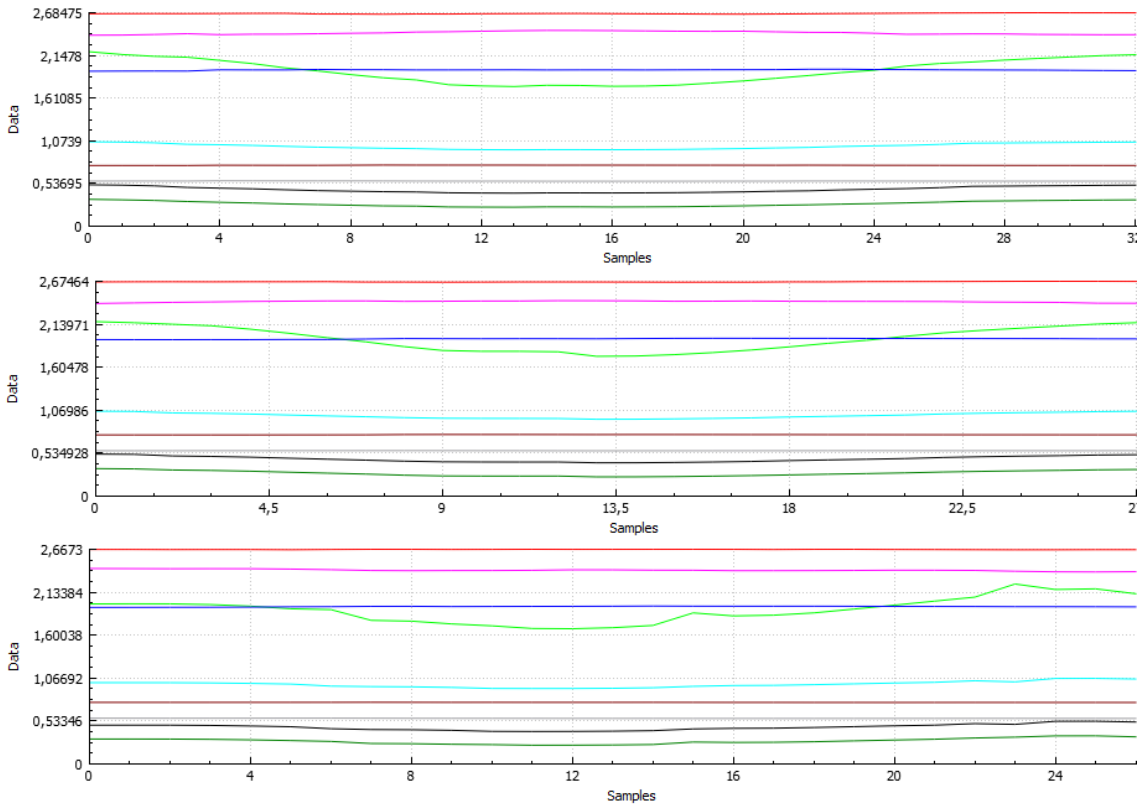


Figure 3.10: Gesture 5 timeseries plot of three samples.

3.2.6 Gesture 6: Sick

Figure 3.11: Sick gesture: Opened hand approaches to forehead making so many moves as syllables have the word.

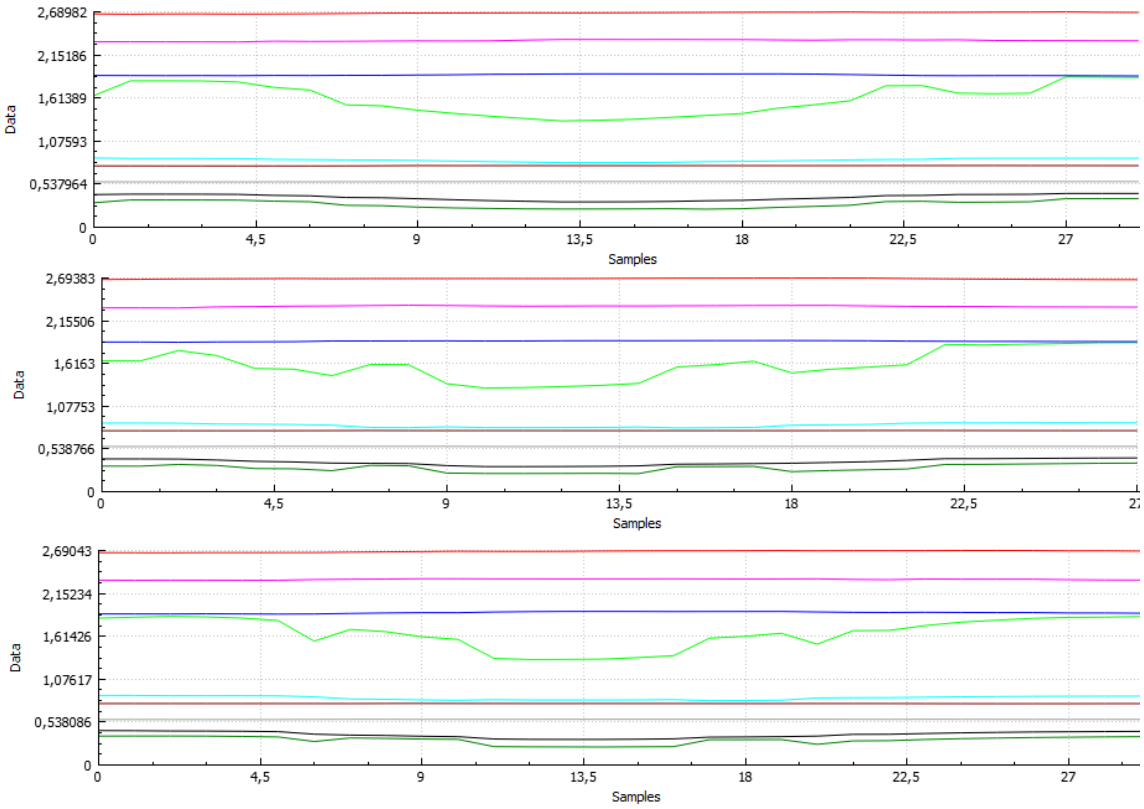


Figure 3.12: Gesture 6 timeseries plot of three samples.

3.2.7 Gesture 7: Clean up

Figure 3.13: Clean up gesture: Closed hand with the knuckles forward. Perform circular movements in the horizontal plane.

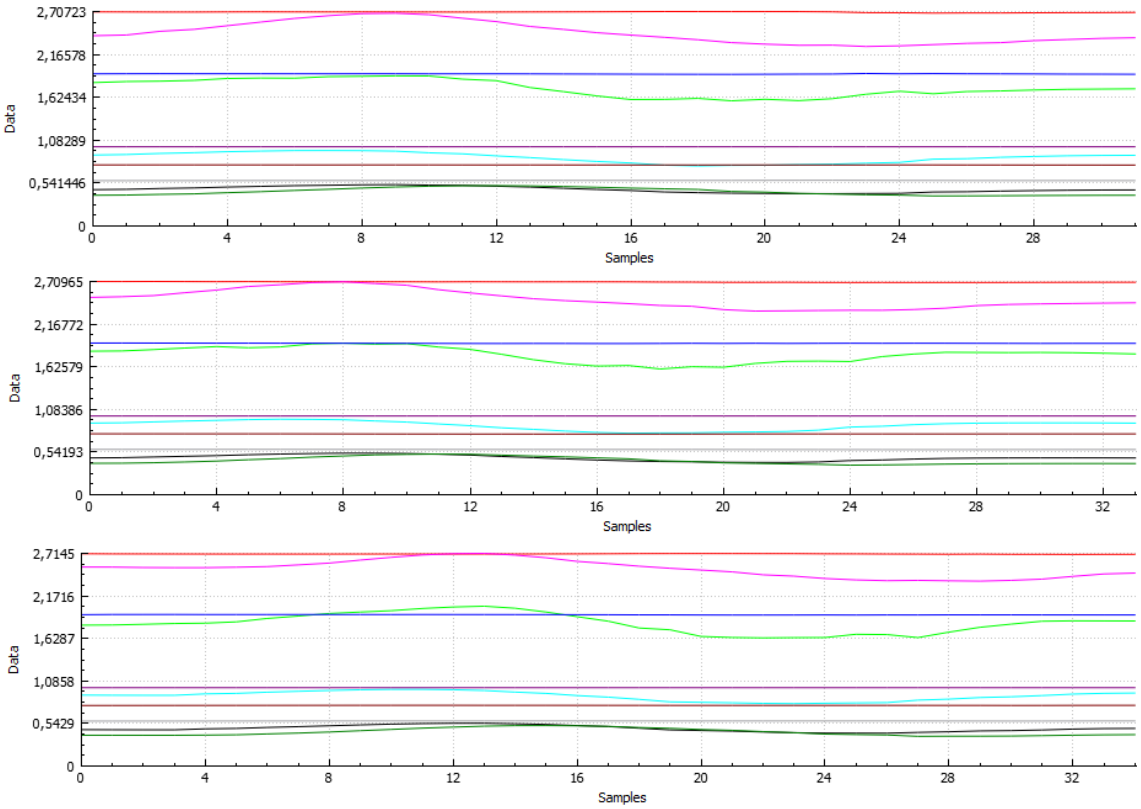


Figure 3.14: Gesture 7 timeseries plot of three samples.

3.2.8 Gesture 8: Mother

Figure 3.15: Mother gesture: Edgeways hand touches the left and right cheek.

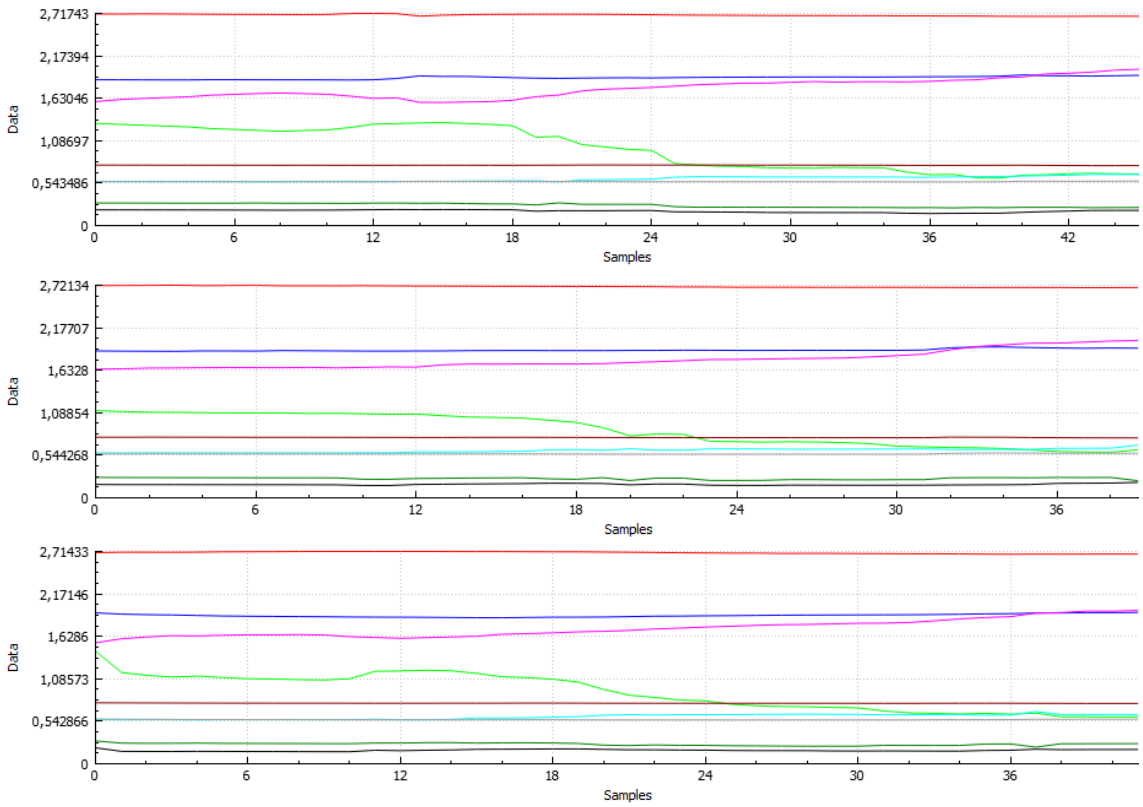


Figure 3.16: Gesture 8 timeseries plot of three samples.

3.2.9 Gesture 9: Father

Figure 3.17: Father gesture: Hand with down palm located in the forehead goes down to the chin.

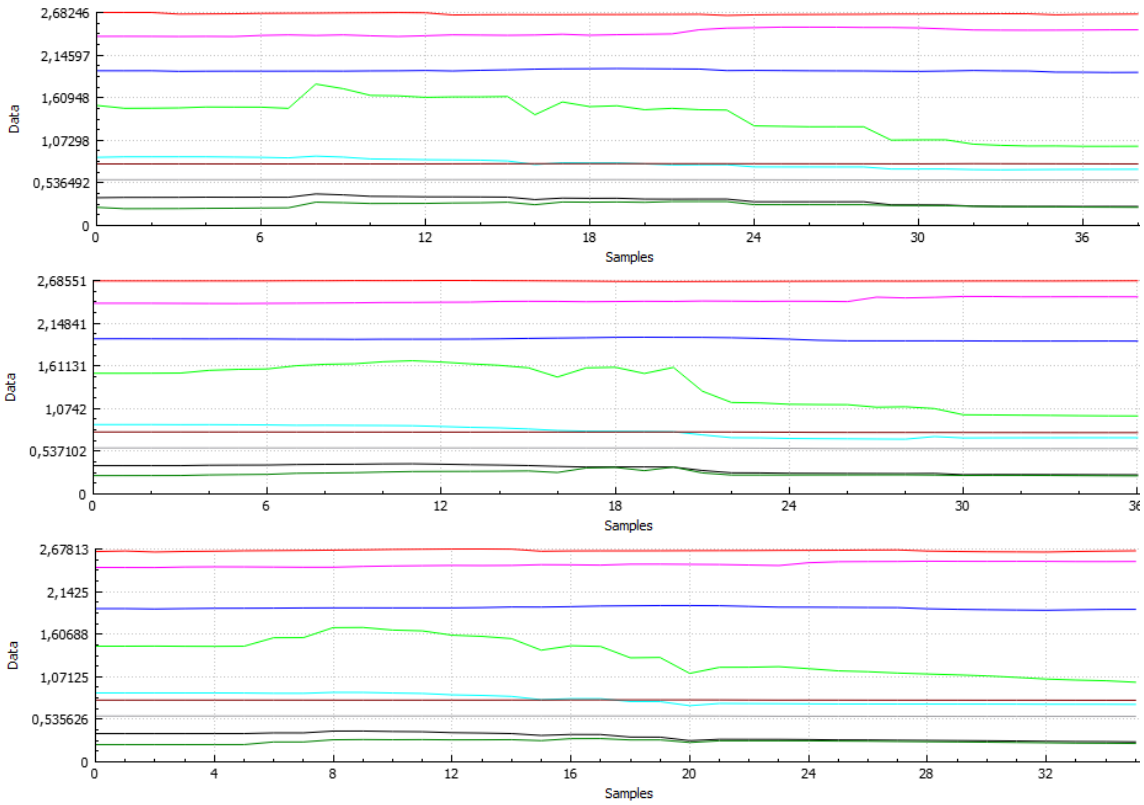


Figure 3.18: Gesture 9 timeseries plot of three samples.

3.2.10 Gesture 10: Want

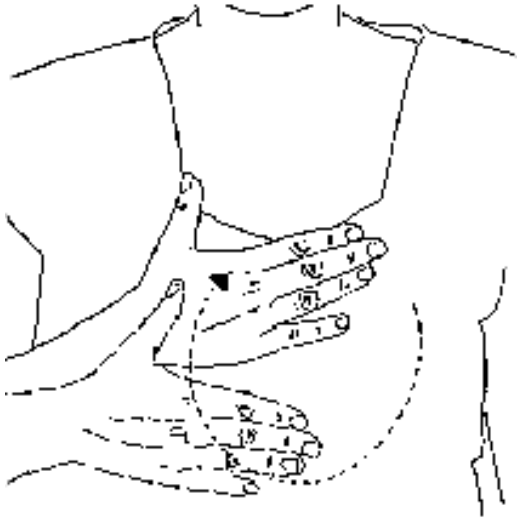


Figure 3.19: Want gesture: Hand with the palm toward the chest, doing a circle live up to this.

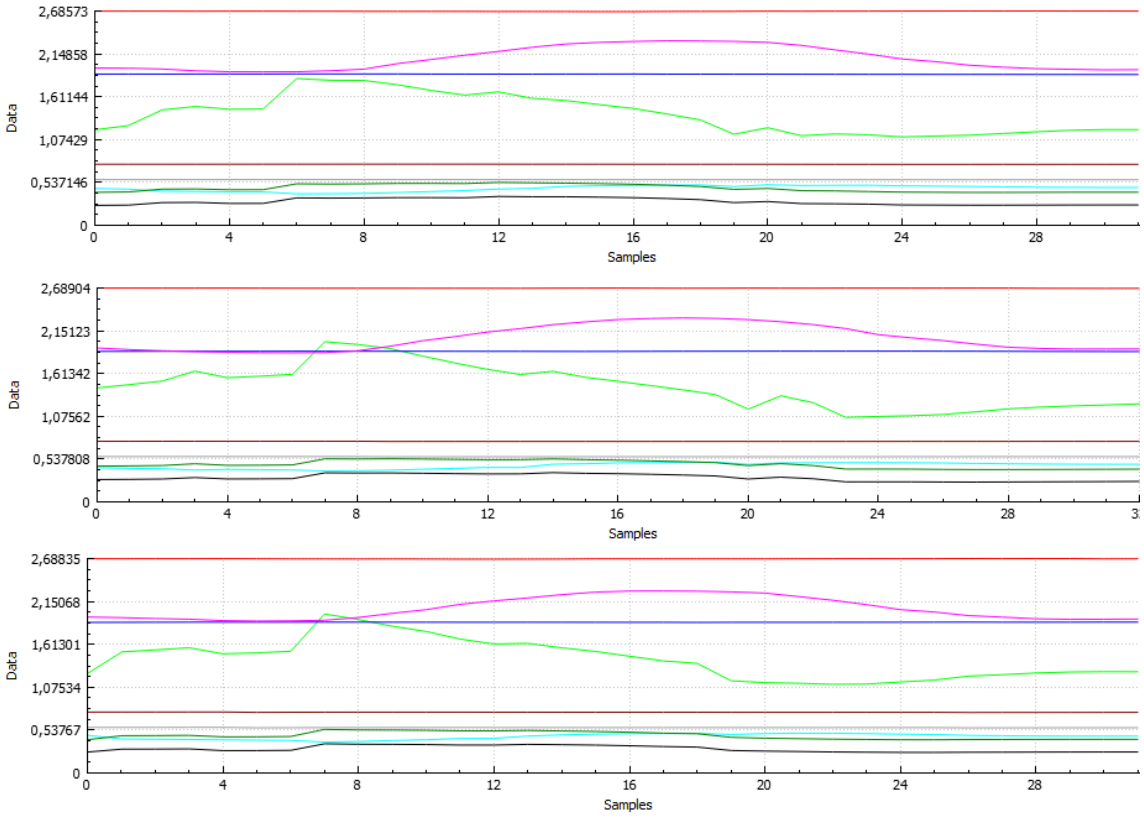


Figure 3.20: Gesture 10 timeseries plot of three samples.

3.2.11 Gesture 11: Dirty

Figure 3.21: Dirty gesture: The edge of the opened palm goes up and down stroking the cheek.

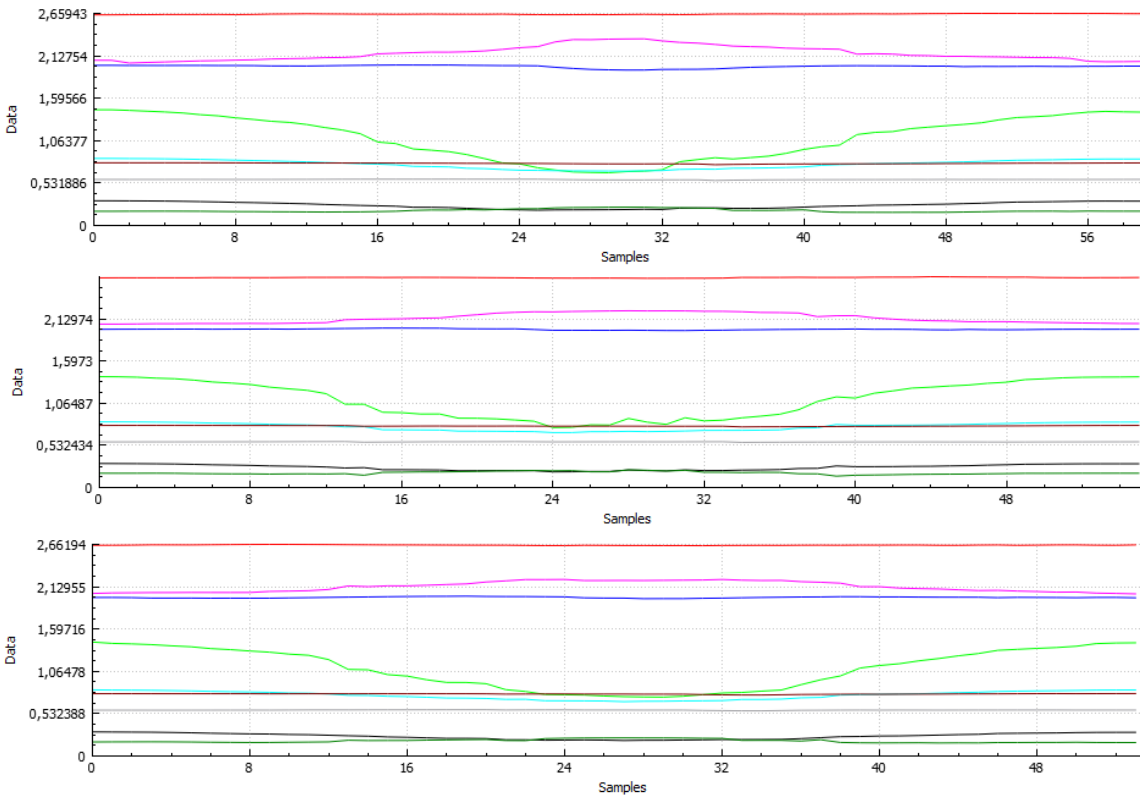


Figure 3.22: Gesture 11 timeseries plot of three samples.

3.3 Gesture recognition pipeline

Our gesture recognition pipeline relies on the *Gesture Recognition Tookit* (GRT) pipeline described in section 2.2.1.1. First of all, a brief analysis of the input data provided by the Kinect sensor was performed. We will also describe in depth, the feature vector resulting from processing the raw input data. All features are associated with a particular gesture, thus justifying its use.

Regarding to the pre-processing and post-processing algorithms, we don't use any technique because of the type and nature of our input data. The majority are incompatible with our vector of features.

3.3.1 Input data

The main input data for our gesture recognition pipeline are the skeletal joints that are relevant for our gesture recognition problem. In order to get the skeletal joints we used Skeletal Tracking functionality of Microsoft Kinect v2 sensor, capable to recognize and track people. Skeletal Tracking is optimized to recognize users standing or sitting, and facing the Kinect sensor; sideways poses provide some challenges regarding the part of the user that is not visible to the sensor. The joints of the body part that are not visible to the sensor field of view, will be approximately inferred by machine learning techniques.

This functionality provides us a number of different body joints. Each joint contains data such as the joint type, position in 3D space (x, y, z) and whether the joint is being tracked. For more details of how Skeletal Tracking works see subsection 2.2.2.1.

The main input data provided by Kinect sensor consists in the location of different joints in the 3D space and relative to the sensor position. With this information we can easily track the position of different body parts, such as hands, arms, chest, shoulders, etc. For our system, the most relevant body joints are shoulders, spine shoulder, head, wrists and elbows. With these joints we can also compute the position of the entire body in the scene, making the system invariant to location. Figure 3.23 shows all skeleton tracked joints. The green marked joints are

the most relevant for our gesture recognition system. They are the following:

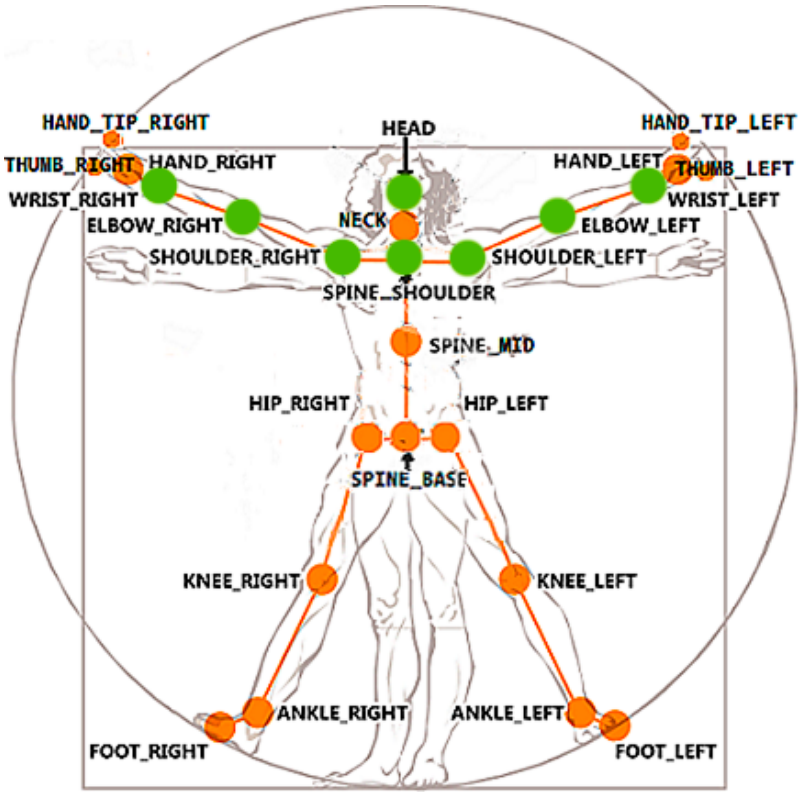


Figure 3.23: Skeleton positions relative to human body. The green joint are the joints we used for the creation of our feature vector (Figure reproduced from `msdn.microsoft.com`)

Figure 3.23 shows all skeleton tracked joints. The green marked joints are the most relevant for our gesture recognition system. They are the following:

- *WRIST_RIGHT*
- *ELBOW_RIGHT*
- *SHOULDER_RIGHT*
- *SPINE_SHOULDER*
- *SHOULDER_LEFT*
- *ELBOW_LEFT*
- *WRIST_LEFT*
- *HEAD*

Moreover, Kinect v2 is able to perform hand tracking by detecting the centroid of the hand and two fingers: index and thumb. It also provides us information about hand state:

- The hand is closed
- The hand is in the lasso state
- Hand state is not tracked
- The hand is open
- The state of he hand is unknown

Processing the information given by these 8 joints, we computed successfully our vector of feature analyzed in the next section.

Among all the possible hand states, we only use two of them in order to avoid the unnecessary overloading of the feature vector. This two hand states were: Closed and Open. The justification for this decision will be discussed in the corresponding section of the involved gestures. By processing the information given by the eight joints listed above beside the two hand states, we successfully created our feature vector analyzed in-depth in the next section.

Listing 3.1 demonstrates how to access and store joints position information in *Vector3D* structure. The code is the same for all the skeleton joints.

Listing 3.1: Extraction of 3-axis position of a joint example.

```
1 Vector3D WristRight;
2 Vector3D ElbowRight;
3
4 WristRight = new Vector3D(joints[JointType.WristRight].Position.X,
5 joints[JointType.WristRight].Position.Y,
6 joints[JointType.WristRight].Position.Z);
7
8 ElbowRight = new Vector3D(joints[JointType.ElbowRight].Position.X,
9 joints[JointType.ElbowRight].Position.Y,
10 joints[JointType.ElbowRight].Position.Z);
```

3.3.2 Feature vector

In pattern recognition and machine learning, a feature vector is an n-dimensional vector of numerical features that represent some object. This is because many machine learning algorithms such as SVM and DTW require a numerical representation of objects.

Our feature vector is obtained by processing the crude data from Kinect sensor. An initial approach was based in the use of skeleton joints position information, without any preprocessing technique. In this case we used wrist and elbow joints. Consequently, our feature vector was composed by 12 features (x,y,z values for four joints: right and left wrists and elbows). With the use of this initial feature vector, the classification results were reasonable, however, in similar gestures, the system prediction wasn't accurate. In order to differentiate between

similar gestures we need to compute concrete features for each particular gesture.

The most important issue is to maximize the available data and obtain a feature vector with relevant information according to our gesture recognition problem. The size of the feature vector should be as much reduced as possible for a faster and more accurate classification process for the new input data.

In our implementation we will consider two types of features. The first one, based on the Euclidean distance (See Subsection 3.3.2.1) between two concrete joints, for example, left and right wrists, in order to find out if the hands are together or not. This feature will be very useful in the case of sleep gesture (See Subsection 3.2.4) where both hands and arms are positioned together.

We will also consider features based on the angle value between joints (See Subsection 3.3.2.2). This is another interesting feature that will allow us to distinguish between various similar gestures where a movement with the arm is performed.

Listing 3.2 shows an example of feature vector used as input for our system. The ClassID label indicates the number of the gesture. In this case its value is 2, so we are dealing with the water gesture. The TimeSeriesLength label indicates the number of feature vectors for this concrete gesture sample. In this particular case we have 58 feature vectors. The feature vector is composed by the following information in the given order: *left elbow angle, right elbow angle, left shoulder angle, right shoulder angle, left wrist to right wrist distance, left wrist to spine shoulder distance, right wrist to spine shoulder distance, left wrist to head distance, right wrist to head distance, left hand closed, right hand closed*.

Listing 3.2: Feature vectors of water gesture

```
1 ************TIME_SERIES************
2 ClassID: 2
3 TimeSeriesLength: 58
4 TimeSeriesData:
5
6 2.61103    1.44735    1.83848    1.87956    0.41871    0.545824
      0.328771    0.738524    0.467313    0    0
7
8 2.61341    1.45511    1.83704    1.87959    0.41879    0.545904
      0.332531    0.738641    0.469459    0    0
9
```

```
10 2.61457    1.4529    1.83696    1.88068    0.419309    0.54582
     0.329701    0.738563    0.46699    0    0
```

The angle values are represented in radians, the distances in meters and if the left or right hand is closed will be indicating with 1 or 0 if open.

3.3.2.1 Features based in Euclidean distance

In mathematics, the Euclidean distance represents the distance between two points in Euclidean space. With this distance, Euclidean space becomes a metric space. In other words, the Euclidean distance between two points p and q is the length of the line segment connecting them ($\overline{pq}$). If $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ are two points in Euclidean n-space then the distance (d) from p to q, or vice versa is given by the Pythagorean formula:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + ... + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{3.1}$$

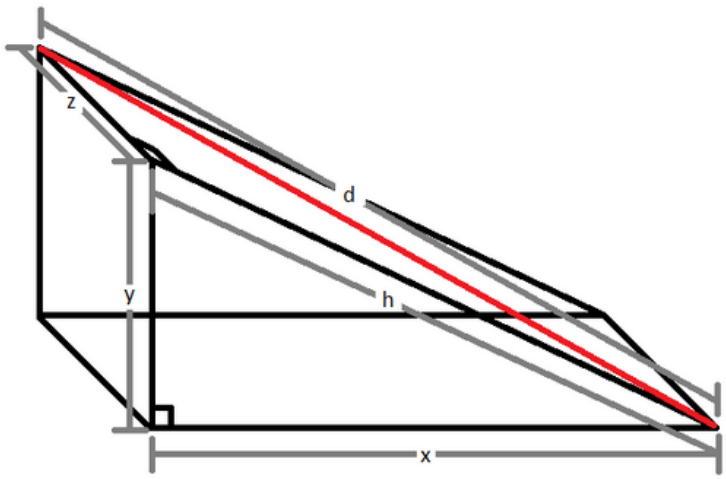


Figure 3.24: Distance between two points in 3D space illustration.

In the Figure 3.24 we can see the illustration of the distance between two points in the 3D space. We will only need to use the Pythagorean theorem twice.

$$d = \sqrt{\sqrt{x^2 + y2^2}^2 + z^2} = \sqrt{x^2 + y^2 + z^2} \tag{3.2}$$

Listing 3.3 shows how we compute the euclidean distance between two *Vector3D* instances. The *Vector3D* structure represents a movement in the 3D space. The Euclidean distance result is expressed in meters.

Listing 3.3: Euclidean distance operation between two Vector3D strucures.

```
1 public double EuclideanDistance(Vector3D vectorA, Vector3D vectorB)
2 {
3 double subtractionX = vectorA.X - vectorB.X;
4 double subtractionY = vectorA.Y - vectorB.Y;
5 double subtractionZ = vectorA.Z - vectorB.Z;
6
7 double sum = Math.Pow(subtractionX, 2) + Math.Pow(subtractionY, 2) +
      Math.Pow(subtractionZ, 2);
8
9 return Math.Sqrt(sum);
10 }
```

Concerning our system implementation and feature vector, we have five euclidean distance features:

- Feature 1: Euclidean distance between left and right wrists
- Feature 2: Euclidean distance between left wrist and spine shoulder
- Feature 3: Euclidean distance between right wrist and spine shoulder
- Feature 4: Euclidean distance between left wrist and head
- Feature 5: Euclidean distance between right wrist and head

In order to analyze each feature one by one, we will perform the gestures with the right hand only.

As we said before, the Euclidean distance between wrists (Feature 1) is very useful in gestures where both wrists are located in similar position (sleep and help gestures, Subsections 3.2.4 and 3.2.1 respectively) or in very different positions (rest of implemented gestures). With this feature we will also be able to distinguish between help and showering (see Subsection 3.2.5) gestures. The feature values in both cases will be very different.

The Euclidean distance between right wrist and spine shoulder (Feature 3) will be very useful in gestures where the hand comes near and moves away the chin (e.g. snack gesture, Subsection 3.2.3). An alternative to the use of the spine shoulder joint, is the neck joint. Using both of them we obtained good results, nevertheless, we opted for the spine shoulder joint for its position that is further

from head joint. Thus, we will be able to better classify the gestures where the hand is moving between spine shoulder and head joints (e.g. father and dirty gesture, Subsection 3.2.9 and 3.2.11 respectively). At the same time, using this feature we will be able to easily distinguish between snack and sick gestures. As we can observe in the Figures 3.6 and 3.12 the movement in the case of both gestures is very similar, nevertheless, the feature 3 values are quite different (see gray line of the graph). This is because of the notable difference in the distance values between the joints. In the snack gesture, the hand is near the spin shoulder, meanwhile in the case of sick gesture, the hand is near the head.

The Euclidean distance between right wrist and head (Feature 5) will be very useful in the same gestures as in the case of Feature 3. When the distance between right wrist and head decreases, the distance between right wrist and spin shoulder increases. Both feature are indirectly proportional. With this feature we will be also be able to distinguish between showering and help gestures (see Subsections 3.2.5 and 3.2.1 respectively). The feature values in both cases will be very different.

For the same gestures made with the left hand, the features would operate in the same way.

3.3.2.2 Features based on angles between joints

The main utility of using these features is to detect motion of the arm while performing the gestures. Concerning our system implementation and feature vector, we have 4 features based on angles between joints:

- Feature 1: Left elbow angle
- Feature 2: Right elbow angle
- Feature 3: Left shoulder angle
- Feature 4: Right shoulder angle

The Features 1 and 2 are useful in order to detect the opening degree of the hand with respect to the shoulder (elbow angle). In this way we will know if the arm is stretched or not. With this feature we will be able to distinguish between father and dirty gestures (See Sections 3.2.9 and 3.2.11 respectively). Both ges-

tures are very similar nevertheless there is a difference in the elbow angle, as we can see in the green line of Figures 3.18 and 3.22 respectively. At the same time, we will be able to distinguish between clean up and want gestures (See Sections 3.2.7 and 3.2.10 respectively).

The Features 3 and 4 are useful in order to detect the opening degree of the elbow with respect to the spin shoulder (shoulder angle). In this way we will know the height of the elbow. If the shoulder angle is greater than 90º, the elbow will be higher than the shoulder. With this information we will be able to distinguish between showering and help gestures (See Subsections 3.2.5 and 3.2.1 respectively) and between snack and sick gestures (See Subsections 3.2.3 and 3.2.6 respectively). In both cases, the position of the elbow with respect to the shoulder is quite different.

The angle between three points in 3D space is easy to compute. Given three points, that contain the x, y and z coordinates: A, B and C. First, we compute $\overrightarrow{BA}$ (v_1) and $\overrightarrow{BC}$ (v_2) vectors as following:

$$v_1 = \{A.x - B.x, A.y - B.y, A.z - B.z\} \tag{3.3}$$

$$v_2 = \{C.x - B.x, C.y - B.y, C.z - B, z\} \tag{3.4}$$

The dot product of v_1 and v_2 is a function of the cosine of the angle between them (it's scaled by the product of their magnitudes). We proceed to normalize both vectors:

$$v_1mag = \sqrt{v_1.x \times v_1.x + v_1.y \times v_1.y + v_1.z \times v_1.z} \tag{3.5}$$

$$v_1norm = \frac{v_1.x}{v_1mag}, \frac{v_1.y}{v_1mag}, \frac{v_1.z}{v_1mag} \tag{3.6}$$

$$v_2mag = \sqrt{v_2.x \times v_2.x + v_2.y \times v_2.y + v_2.z \times v2.z} \tag{3.7}$$

$$v_2norm = \frac{v_2.x}{v_2mag}, \frac{v_2.y}{v_2mag}, \frac{v_2.z}{v_2mag} \tag{3.8}$$

Then we calculate the dot product,

$$result = v_1norm.x \times v_2norm.x + v_1norm.y \times v_2norm.y + v_1norm.z \times v_2norm.z, \tag{3.9}$$

and finally we recover the angle in radians

$$angle = \arccos(result). \tag{3.10}$$

Listing 3.4 shows the implementation of the process described above.

Listing 3.4: Calculating the angle between two vectors

```
1  public double AngleBetweenTwoVectors(Vector3D vectorA, Vector3D
       vectorB)
2  {
3  double dotProduct = 0.0;
4  vectorA.Normalize();
5  vectorB.Normalize();
6  dotProduct = Vector3D.DotProduct(vectorA, vectorB);
7
8  //Conversion from rad to deg, add: / Math.PI * 180
9  return (double)Math.Acos(dotProduct);
10 }
```

3.3.3 Training and classification with dynamic time warping

Dynamic time warping (DTW) is a well-known technique to find an optimal alignment between two given (time-dependent) sequences under certain restrictions (Figure 3.25) [44]. These sequences are warped in a nonlinear fashion to match each other. Originally, DTW was implemented in order to compare different speech patterns in automatic speech recognition. Its use has spread to fields such as data mining and information retrieval where has been successfully applied to automatically cope with time deformations and different speeds associated with time-dependent data. For instance, similarities in gesture patterns could be detected using DTW, even if one person is performing the gesture faster than the other, or if there were accelerations and decelerations during the course of an observation. Computing the DTW requires $\mathcal{O}(N^2)$ in general. There are fast techniques for computing DTW such as SparseDTW and the FastDTW, nevertheless we used the classical DTW implementation.

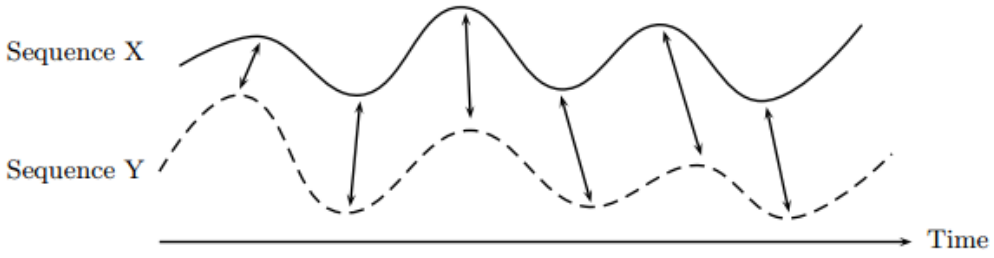


Figure 3.25: Time alignment of two time-dependent sequences. Aligned points are indicated by the arrows (Figure reproduced from [44]).

3.4 Pipeline performance study

In order to validate our gesture recognition system we used 5-fold cross validation. The Gesture Recognition Toolkit doesn't implement this validation technique in the GUI, nevertheless we implemented our own solution using the library. Listing 3.5 shows the implementation of our custom k-fold validation technique.

Listing 3.5: 5-fold cross validation custom implementation

```
1  for (int fold = 0; fold < numFolds; fold++)
2  {
3  //You can then ask for the training data for fold k
4  LabelledTimeSeriesClassificationData foldTrainingData =
       trainingData.getTrainingFoldData(fold);
5
6  pipeline.train(foldTrainingData);
7
8  //And the test data for fold k
9  LabelledTimeSeriesClassificationData foldTestingData = trainingData
       .getTestFoldData(fold);
10
11 //Test the classification accuracy of the trained pipeline
12 bool testSuccess = pipeline.test(foldTestingData);
13 if (!testSuccess) cout << Testing Error << endl;
14
15 cout << endl;
16
17 //You can then get then get the accuracy of how well the pipeline
       performed with the test data
18
19 cout << Fold:  << fold + 1 << endl;
20 cout << Accuracy:  << pipeline.getTestAccuracy() << endl;
21 cout << Training Samples Number:  << pipeline.getNumTrainingSamples
       () << endl;
22 cout << Test Samples Number:  << pipeline.getNumTestSamples() <<
       endl;
23 cout << Training Time:  << pipeline.getTrainingTime() << endl;
24 cout << Test Time:  << pipeline.getTestTime() << endl;
```

```
25  cout << endl;
26
27  classLabels = pipeline.getClassLabels();
28
29  for (int gesture = 0; gesture < classLabels.size(); gesture++)
30  {
31  cout << Gesture:  << classLabels[gesture] << endl;
32  double fMeasure = pipeline.getTestFMeasure(classLabels[gesture]);
33  cout << F-Measure:  << fMeasure << endl;
34
35  double precision = pipeline.getTestPrecision(classLabels[gesture]);
36  cout << Precision:  << precision << endl;
37
38  double recall = pipeline.getTestRecall(classLabels[gesture]);
39  cout << Recall:  << recall << endl;
40  cout << endl;
41  }
42  getchar();
43  }
```

We performed two series of tests with different gestures datasets. The differences between the datasets are basically regarding to the number of samples. One dataset has 15 samples by each gesture and the other has 30 samples by each gesture. As we will see, there are differences in the obtained results. The dataset with 15 samples by gesture obtains higher accuracy rates. Nevertheless using the system in order to predict new input data, the classification process is not so accurate as in the results obtained with the cross validation technique. This is because of the reduced number of samples. When a new gesture is performed by other person the matching probability with one of the 15 samples is lower than when using 30 samples by gesture. As we can see, the accuracy rate should be directly proportional with the number of samples by each gesture. This fact doesn't always have to fulfill because when we introduce more gestures samples, the probability that samples of different gesture be similar is higher.

As we observe, the accuracy rates for the dataset with 30 samples by gesture are worst than the dataset with 15 samples. This is because the error rate in the classification is directly proportional with the number of samples used for training.

3.4.1 Dataset with 15 samples by gesture

5-fold cross validation results using a dataset with 15 samples by each one of the eleven implemented gestures.

Fold 1 Accuracy 96.875%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	1	1	1	1	1	1	1	0.857	1	0.8
Precision	1	1	1	1	1	1	1	1	0.75	1	1
Recall	1	1	1	1	1	1	1	1	1	1	0.667

Fold 2 Accuracy 96.5517%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	1	1	1	1	1	1	1	0.857	1	0.8
Precision	1	1	1	1	1	0.667	1	1	1	1	1
Recall	1	1	1	1	1	1	1	1	1	1	0.667

Fold 3 Accuracy 88.4615%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	0.857	0.8	1	1	0.667	1	1	1	1	IND
Precision	1	0.75	1	1	1	0.667	1	1	1	1	0
Recall	1	1	0.667	1	1	1	1	1	1	1	0

Fold 4 Accuracy 95.8333%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	1	1	1	1	0.667	1	1	0.857	1	1
Precision	1	1	1	1	1	1	1	1	0.75	1	1
Recall	1	1	1	1	1	0.6	1	1	1	1	1

Fold 5 Accuracy 95.2381%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	1	1	1	1	0.8	1	1	1	1	0.667
Precision	1	1	1	1	1	0.667	1	1	1	1	1
Recall	1	1	1	1	1	1	1	1	1	1	0.5

3.4.2 Dataset with 30 samples by gesture

5-fold cross validation results using a dataset with 30 samples by each one of the eleven implemented gestures.

Fold 1 Accuracy 71.4286%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	0.889	0.833	1	0.571	0.4	0.75	0.667	0.889	0.75	0.4
Precision	1	1	0.833	1	1	0.25	1	1	1	1	1
Recall	1	0.8	0.833	1	0.4	1	0.6	0.5	0.8	0.6	0.25

Fold 2 Accuracy 77.7778%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	0.75	0.8	1	0.333	0.526	0.889	0.8	0.667	1	0.857
Precision	1	1	0.8	1	1	0.357	1	0.8	0.75	1	1
Recall	1	0.6	0.8	1	0.2	1	0.8	0.8	0.6	1	0.75

Fold 3 Accuracy 72.2222%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	0.571	1	0.889	0.571	0.417	1	0.8	0.571	0.75	0.667
Precision	1	1	1	1	1	0.263	1	0.8	1	1	1
Recall	1	0.4	1	0.8	0.4	1	1	0.8	0.4	0.6	0.5

Fold 4 Accuracy 94.2308%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	0.889	1	1	0.769	1	1	0.857	1	1	0.8
Precision	1	1	1	1	1	0.625	1	1	1	1	1
Recall	1	0.8	1	1	1	1	1	1	0.75	1	0.667

Fold 5 Accuracy 83.3333%

Gesture	1	2	3	4	5	6	7	8	9	10	11
F-Measure	1	0.667	1	1	0.667	0.555	1	1	0.857	0.888	0.5
Precision	1	1	1	1	1	0.385	1	1	1	1	1
Recall	1	0.5	1	1	0.5	1	1	1	0.75	0.8	0.333

Chapter 4

Conclusions

This chapter discusses the main conclusions extracted from the work presented in this document. The chapter is divided into three different sections: Section 4.1 presents and discusses the final conclusions of the work presented in this Bachelor's Thesis. Finally, Section 4.2 presents future works: open problems and research topics that remain for future research.

4.1 Conclusions

In this work, we have implemented a pipeline for a gesture recognition system using the Gesture Recognition Toolkit (GRT). The developed system is able to recognize eleven gesture from the Schaeffer sign language in cluttered scenes and with a real-time execution. The experiments that were carried out proved the accuracy of the proposal and validated our system. The system development process was a real challenge specially when designing our feature vector and analyzing each gesture one by one. At the same time, the experimentation was difficult due to the lack of a ground truth. During the development, we found a pretty large number of bugs in the Gesture Recognition Toolkit and we reported them to the developer. At the same time, we noticed that the GRT is an experimental project, nevertheless we have used it successfully. Its installation was quite complicated and we needed to manage different compilation errors.

From a personal point of view, I feel that project has allowed me to show

off the knowledge acquired during the last four years. This project was quite ambitious but I am proud of the achieved results. The project was a real challenge for me.

4.2 Future work

There are a great number of improvements and ideas to easily extend or improve this project. However, due to the lack of time, we limited the scope of the project and build it regarding more realistic goals.

For future work we can extend our pipeline implementation to static gesture in order to determine the posture of the hand in the space at the same time we recognize dynamic gestures. We can easily detect if a gesture is static or dynamic calculating the variation of skeleton joints position between two consecutive frames. If a considerable variation exists in the joints position, a movement was performed so we will pretend to recognize dynamic gestures. If there is no variation in joints position, we will use a static gesture approach. In order to recognize both static and dynamic gestures at the same time we can rely on a parallel implementation on CPU using threads.

At the same time we can parallelize our gesture recognition pipeline over GPU using CUDA in order to achieve a real-time performance and a faster training and classification process.

The implemented gestures in this project were extracted from Schaeffer sign language. Their executions are quite distinguishable since the movements of most of them are enough different. Nevertheless, we can extend our project to the recognition of the American Sign Language [5] that is much more complex. In this case a hand pose estimation will be absolutely necessary in order to distinguish between gestures.

Appendix A

Gesture Recognition Tooklit build and linking instructions

A.1 Building instructions

First, we will need to download the CMake Windows installer and a compiler that supports C++ 11 (such as Visual Studio 2013 and 2015).

To build the GRT library on Windows as a static library and compile the examples:

- We proceed to create a temporary build folder in the same directory of GRT.
- With CMake GUI we select the build folder created and the folder of the source code with the CMakeLists.
- We will also enable the option -DBUILD_STATIC_LIB=ON in order to build static library. This first steps will generate project files for all the GRT classes and examples in the temporary build directory.
- Open the main project file with Visual Studio 2013 or 2015, which should be called ALL_BUILD.vcxproj.
- Use Visual Studio to compile the GRT library and examples
- If this is successfull, it should build the GRT as a static library (grt.lib) and build all the GRT examples
- We can now either run the pre-built GRT examples, or use the GRT static library in your custom projects

A.2 Linking instructions

After building the GRT as a static library, you can build a new custom project and link against the GRT library by:

- Create a new Visual Studio C++ project.
- Add some custom code to your project.
- Select the project in the Visual Studio 'Solution' window, right click and select 'Properties'.
- Select the 'C/C++', 'General', 'Additional Include Directories' and add the path to the folder containing the GRT source code and main GRT header (GRT.h).
- Select the 'Linker','General', 'Additional Library Directories' and add the path to the folder containing the GRT static library (grt.lib, this should be in a folder called Debug in your temporary build directory).
- Select the 'Linker','Input', 'Additional Dependencies' and add the GRT static library (grt.lib).
- Click 'Apply', you should now be able to build your custom project and link against the GRT.

Bibliography

[1] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly, "Vision-based hand pose estimation: A review," *Computer Vision and Image Understanding*, vol. 108, no. 1–2, pp. 52 – 73, 2007, special Issue on Vision for Human-Computer Interaction. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314206002281

[2] H. Hasan and S. Abdul-Kareem, "Static hand gesture recognition using neural networks," *Artificial Intelligence Review*, vol. 41, no. 2, pp. 147–181, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10462-011-9303-1

[3] S. Mitra and T. Acharya, "Gesture recognition: A survey," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 3, pp. 311–324, May 2007.

[4] S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 1–54, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10462-012-9356-9

[5] V. Pavlovic, R. Sharma, and T. Huang, "Visual interpretation of hand gestures for human-computer interaction: a review," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 7, pp. 677–695, Jul 1997.

[6] B. Schaeffer, A. Musil, and G. Kollinzas, *Total communication: A signed speech program for nonverbal children*. Research, 1985.

[7] A. Hollister, W. L. Buford, L. M. Myers, D. J. Giurintano, and A. Novick, "The axes of rotation of the thumb carpometacarpal joint," *Journal of Orthopaedic Research*, vol. 10, no. 3, pp. 454–460, 1992. [Online]. Available: http://dx.doi.org/10.1002/jor.1100100319

[8] P. Premaratne, *Human computer interaction using hand gestures*. Springer Science & Business Media, 2014.

[9] A. Erol, G. Bebis, M. Nicolescu, R. Boyle, and X. Twombly, "A review on vision-based full dof hand motion estimation," in *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, June 2005, pp. 75–75.

[10] D. Sturman and D. Zeltzer, "A survey of glove-based input," *Computer Graphics and Applications, IEEE*, vol. 14, no. 1, pp. 30–39, Jan 1994.

[11] M. Massachusetts Institute of Technology. (2009. Accesed August 5, 2015.) Mit acceleglove. http://www.technologyreview.com/article/414021/open-source-data-glove/.

[12] R. Y. Wang and J. Popović, "Real-time hand-tracking with a color glove," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.

[13] J. Davis and M. Shah, "Recognizing hand gestures," in *Computer Vision — ECCV '94*, ser. Lecture Notes in Computer Science, J.-O. Eklundh, Ed. Springer Berlin Heidelberg, 1994, vol. 800, pp. 331–340. [Online]. Available: http://dx.doi.org/10.1007/3-540-57956-7_37

[14] Y. Iwai, K. Watanabe, Y. Yagi, and M. Yachida, "Gesture recognition using colored gloves," in *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, vol. 1, Aug 1996, pp. 662–666 vol.1.

[15] L. Lamberti and F. Camastra, "Real-time hand gesture recognition using a color glove," in *Image Analysis and Processing – ICIAP 2011*, ser. Lecture Notes in Computer Science, G. Maino and G. Foresti, Eds. Springer Berlin Heidelberg, 2011, vol. 6978, pp. 365–373. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24085-0_38

[16] J. Rehg and T. Kanade, "Digiteyes: vision-based hand tracking for human-computer interaction," in *Motion of Non-Rigid and Articulated Objects, 1994., Proceedings of the 1994 IEEE Workshop on*, Nov 1994, pp. 16–22.

[17] T. Darrell and A. Pentland, "Space-time gestures," in *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR '93., 1993 IEEE Computer Society Conference on*, Jun 1993, pp. 335–340.

[18] A. Utsumi, T. Miyasato, and F. Kishino, "Multi-camera hand pose recognition system using skeleton image," in *Robot and Human Communication, 1995. RO-MAN'95 TOKYO, Proceedings., 4th IEEE International Workshop on*, Jul 1995, pp. 219–224.

[19] W. T. Freeman and M. Roth, "Orientation histograms for hand gesture recognition," in *International workshop on automatic face and gesture recognition*, vol. 12, 1995, pp. 296–301.

[20] W. T. Freeman, "Dynamic and static hand gesture recognition through low-level image analysis," Sep. 26 1995, uS Patent 5,454,043.

[21] Y. Cui, D. Swets, and J. Weng, "Learning-based hand sign recognition using shoslif-m," in *Computer Vision, 1995. Proceedings., Fifth International Conference on*, Jun 1995, pp. 631–636.

[22] N. Shimada, Y. Shirai, Y. Kuno, and J. Miura, "Hand gesture estimation and model refinement using monocular camera-ambiguity limitation by inequality constraints," in *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, Apr 1998, pp. 268–273.

[23] J. Segen and S. Kumar, "Gesture vr: Vision-based 3d hand interace for spatial interaction," in *Proceedings of the Sixth ACM International Conference on Multimedia*, ser. MULTIMEDIA '98. New York, NY, USA: ACM, 1998, pp. 455–464. [Online]. Available: http://doi.acm.org/10.1145/290747.290822

[24] A. Utsumi and J. Ohya, "Multiple-hand-gesture tracking using multiple cameras," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 1, 1999, pp. –478 Vol. 1.

[25] F.-S. Chen, C.-M. Fu, and C.-L. Huang, "Hand gesture recognition using a real-time tracking method and hidden markov models," *Image and Vision Computing*, vol. 21, no. 8, pp. 745 – 758, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885603000702

[26] D. J. Sturman, "Whole-hand input," Ph.D. dissertation, Massachusetts Institute of Technology, 1991.

[27] M. Imaging. (2006. Accesed February 19, 2015.) Mesa Imaging. http://www.mesa-imaging.ch/.

[28] J. Garcia and Z. Zalevsky, "Range mapping using speckle decorrelation," Oct. 7 2008, uS Patent 7,433,024. [Online]. Available: http://www.google.com.ar/patents/US7433024

[29] N. Gillian and J. A. Paradiso, "The gesture recognition toolkit," *Journal of Machine Learning Research*, vol. 15, pp. 3483–3487, 2014. [Online]. Available: http://jmlr.org/papers/v15/gillian14a.html

[30] M. Nicholas Gillian, Massachusetts Institute of Technology. (December 14, 2013.) Gesture recognition toolkit wiki. http://www.nickgillian.com/wiki/pmwiki.php/GRT/GestureRecognitionToolkit.

[31] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1297–1304. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2011.5995316

[32] R. Open Source Robotics Fundation. (2011. Accesed February 19, 2015.) TurtleBot robot. http://www.turtlebot.com/.

[33] M. Morana, "3d scene reconstruction using kinect," in *Advances onto the Internet of Things*, ser. Advances in Intelligent Systems and Computing, S. Gaglio and G. Lo Re, Eds. Springer International Publishing, 2014, vol. 260, pp. 179–190. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-03992-3_13

[34] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '11. New York, NY, USA: ACM, 2011, pp. 559–568. [Online]. Available: http://doi.acm.org/10.1145/2047196.2047270

[35] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *Int. J. Rob. Res.*, vol. 31, no. 5, pp. 647–663, Apr. 2012. [Online]. Available: http://dx.doi.org/10.1177/0278364911434148

[36] K. Lai, L. Bo, X. Ren, and D. Fox, "Sparse distance learning for object recognition combining rgb and depth information," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 4007–4013.

[37] Y. Li, "Hand gesture recognition using kinect," in *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on*, June 2012, pp. 196–199.

[38] Microsoft. (2010. Accesed February 22, 2015., Mar) Primesense supplies 3-d-sensing technology to "project natal" for xbox 360. http://news.microsoft.com/2010/03/31/primesense-supplies-3-d-sensing-technology-to-project-natal-for-xbox-360/.

[39] B. Freedman, A. Shpunt, M. Machline, and Y. Arieli, "Depth mapping using projected patterns," Apr. 3 2012, uS Patent 8,150,142.

[40] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, p. 1437, 2012. [Online]. Available: http://www.mdpi.com/1424-8220/12/2/1437

[41] D. L. Lau. (November 27, 2013.) The science behind kinect. http://lau.engineering.uky.edu/2013/11/27/the-science-behind-kinect/.

[42] T. Butkiewicz, "Low-cost coastal mapping using kinect v2 time-of-flight cameras," in *Oceans - St. John's, 2014*, Sept 2014, pp. 1–9.

[43] Y. Zhang and L. Wu, "Crop classification by forward neural network with adaptive chaotic particle swarm optimization," *Sensors*, vol. 11, no. 5, pp. 4721–4743, 2011.

[44] "Dynamic time warping (dtw)," in *Encyclopedia of Genetics, Genomics, Proteomics and Informatics*. Springer Netherlands, 2008, pp. 570–570. [Online]. Available: http://dx.doi.org/10.1007/978-1-4020-6754-9_4969