Universidade do Minho
Escola de Engenharia

Paulo José de Albuquerque C. Trigueiros

Hand Gesture Recognition System based in Computer Vision and Machine Learning : Applications on Human-Machine Interaction

Outubro de 2013

UMinho | 2013

Paulo José de Albuquerque C. Trigueiros

Hand Gesture Recognition System based in Computer Vision
and Machine Learning : Applications on Human-Machine Interaction

Universidade do Minho
Escola de Engenharia

Paulo José de Albuquerque C. Trigueiros

Hand Gesture Recognition System based in Computer Vision and Machine Learning : Applications on Human-Machine Interaction

Tese de Doutoramento
Electronic and Computer Engineering

Trabalho efectuado sob a orientação do
Professor Doutor Fernando Ribeiro

e co-orientação do
Professor Doutor Luís Paulo Reis

Outubro de 2013

Declaration

Name: Paulo José de Albuquerque C. Trigueiros
E-mail: ptrigueiros@gmail.com
Phone number: (+351) 938 610 888
Passport number: 77776094

Doctoral thesis title:

Hand Gesture Recognition System based in Computer Vision and Machine Learning : Applications on Human-Machine Interaction

Supervisor: Professor Fernando Ribeiro
Co-supervisor: Professor Doutor Luís Paulo Reis

Year of conclusion: 2013

Area of knowledge: Electronic and Computer Engineering

THE INTEGRAL REPRODUCTION OF THIS DISSERTATION IS ONLY AUTHORIZED FOR RESEARCH EFFECTS, AFTER WRITTEN DECLARATION OF THE INTERESTED PARTY, TO WHICH IT PLEDGES TO COMPLY.

University of Minho, October 2010

Paulo Trigueiros

Acknowledgement

First of all I have to thank my advisors, Professor Fernando Ribeiro and Professor Luís Paulo Reis, for their friendship, their omnipresent criticism to my work, their constant availability to clarify my doubts and my inexperience, and their guidance during this difficult process of writing a dissertation. I will never have words to thank your persistence with me. A special thanks to Professor Fernando Ribeiro for the wise words that allowed me to continue forward in difficult times during this work.

I would also like to thank Professor Gil Lopes for all the ideas discussed during coffee breaks or during lunch, for his true friendship, for his ability to focus on issues that were very important for the proper development of the work and for his constant sympathy in difficult times.

I would like to thank to all the friends that support and encourage me during this difficult journey. I would also like to thank all the people who volunteered to test all the developed prototypes during this work and that also participated in the data collection phase, which was of utmost importance for the experiments carried out allowing to successfully complete this thesis.

I would like to thanks my parents for life and for all the opportunities they were able to provide me during my life time.

And, finally, there are no words to express the gratitude to my family, for their patience and dedication, for their love, especially my wife, for the constant words of encouragement and for being always by my side. This thesis is also yours. My wife and my sons are my reason for living. I love you all.

Paulo Trigueiros

Resumo

Sendo uma forma natural de interação homem-máquina, o reconhecimento de gestos implica uma forte componente de investigação em áreas como a visão por computador e a aprendizagem computacional. O reconhecimento gestual é uma área com aplicações muito diversas, fornecendo aos utilizadores uma forma mais natural e mais simples de comunicar com sistemas baseados em computador, sem a necessidade de utilização de dispositivos extras. Assim, o objectivo principal da investigação na área de reconhecimento de gestos aplicada à interacção homem-máquina é o da criação de sistemas, que possam identificar gestos específicos e usá-los para transmitir informações ou para controlar dispositivos. Para isso as interfaces baseados em visão para o reconhecimento de gestos, necessitam de detectar a mão de forma rápida e robusta e de serem capazes de efetuar o reconhecimento de gestos em tempo real. Hoje em dia, os sistemas de reconhecimento de gestos baseados em visão são capazes de trabalhar com soluções específicas, construídos para resolver um determinado problema e configurados para trabalhar de uma forma particular. Este projeto de investigação estudou e implementou soluções, suficientemente genéricas, com o recurso a algoritmos de aprendizagem computacional, permitindo a sua aplicação num conjunto alargado de sistemas de interface homem-máquina, para reconhecimento de gestos em tempo real. A solução proposta, *Gesture Learning Module Architecture* (GeLMA), permite de forma simples definir um conjunto de comandos que pode ser baseado em gestos estáticos e dinâmicos e que pode ser facilmente integrado e configurado para ser utilizado numa série de aplicações. É um sistema de baixo custo e fácil de treinar e usar, e uma vez que é construído unicamente com bibliotecas de código. As experiências realizadas permitiram mostrar que o sistema atingiu uma precisão de 99,2% em termos de reconhecimento de gestos estáticos e uma precisão média de 93,7% em termos de reconhecimento de gestos dinâmicos. Para validar a solução proposta, foram implementados dois sistemas completos. O primeiro é um sistema em tempo real capaz de ajudar um árbitro a arbitrar um jogo de futebol robótico. A solução proposta combina um sistema de reconhecimento de gestos baseada em visão com a definição de uma linguagem formal, *o CommLang Referee*, à qual demos a designação de *Referee Command Language Interface System* (ReCLIS). O sistema identifica os comandos baseados num conjunto de gestos estáticos e dinâmicos executados pelo árbitro, sendo este posteriormente enviado para um interface de computador que transmite a respectiva informação para os robôs. O segundo é um sistema em tempo real capaz de interpretar um subconjunto da Linguagem Gestual Portuguesa. As experiências demonstraram que o sistema foi capaz de reconhecer as vogais em tempo real de forma fiável. Embora a solução implementada apenas tenha sido treinada para reconhecer as cinco vogais, o sistema é facilmente extensível para reconhecer o resto do alfabeto. As experiências também permitiram mostrar que a base dos sistemas de interação baseados em visão pode ser a mesma para todas as aplicações e, deste modo facilitar a sua implementação. A solução proposta tem ainda a vantagem de ser suficientemente genérica e uma base sólida para o desenvolvimento de sistemas baseados em reconhecimento gestual que podem ser facilmente integrados com qualquer aplicação de interface homem-máquina. A linguagem formal de definição da interface pode ser redefinida e o sistema pode ser facilmente configurado e treinado com um conjunto de gestos diferentes de forma a serem integrados na solução final.

Abstract

Hand gesture recognition is a natural way of human computer interaction and an area of very active research in computer vision and machine learning. This is an area with many different possible applications, giving users a simpler and more natural way to communicate with robots/systems interfaces, without the need for extra devices. So, the primary goal of gesture recognition research applied to Human-Computer Interaction (HCI) is to create systems, which can identify specific human gestures and use them to convey information or controlling devices. For that, vision-based hand gesture interfaces require fast and extremely robust hand detection, and gesture recognition in real time.

Nowadays, vision-based gesture recognition systems are able to work with specific solutions, built to solve one particular problem and configured to work in a particular manner. This research project studied and implemented solutions, generic enough, with the help of machine learning algorithms, allowing its application in a wide range of human-computer interfaces, for real-time gesture recognition.

The proposed solution, *Gesture Learning Module Architecture* (GeLMA), allows the definition in a simple way of a set of commands that can be based on static and dynamic gestures and that can be easily integrated and configured to be used in a number of applications. It is easy to train and use, and since it is mainly built with open source libraries it is also an inexpensive solution. Experiments carried out showed that the system achieved an accuracy of 99.2% in terms of hand posture recognition and an average accuracy of 93,72% in terms of dynamic gesture recognition. To validate the proposed framework, two systems were implemented. The first one is an online system able to help a robotic soccer game referee judge a game in real time. The proposed solution combines a vision-based hand gesture recognition system with a formal language definition, the *Referee CommLang*, into what is called the *Referee Command Language Interface System* (ReCLIS). The system builds a command based on system-interpreted static and dynamic referee gestures, and is able to send it to a computer interface which can then transmit the proper commands to the robots. The second one is an online system able to interpret the Portuguese Sign Language. The experiments showed that the system was able to reliably recognize the vowels in real-time. Although the implemented solution was only trained to recognize the five vowels, it is easily extended to recognize the rest of the alphabet. These experiments also showed that the core of vision-based interaction systems can be the same for all applications and thus facilitate its implementation. The proposed framework has the advantage of being generic enough and a solid foundation for the development of hand gesture recognition systems that can be integrated in any human-computer interface application. The interface language can be redefined and the system can be easily configured to train different sets of gestures that can be easily integrated into the final solution.

Contents

Acknowledgement .... **iii**

Resumo .... **v**

Abstract .... **vii**

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 3

1.3 Approach 4

1.4 Contributions 5

1.5 Document Structure 6

2 Methodologies and State of the Art **9**

2.1 Introduction 9

2.2 Background 10

2.2.1 Feature Selection and Extraction 10

2.2.2 Centroid Distance Signature 11

2.2.3 Histogram of Oriented Gradients (HoG) 12

2.2.4 Local Binary Patterns 14

2.2.5 Fourier Descriptors 17

2.2.6 The Shi-Tomasi Corner Detector 19

2.2.7 Related Work 19

2.3 Machine Learning and Classification 27

2.3.1 k-Nearest Neighbour (k-NN) 28

2.3.2 Artificial Neural Networks (ANN) 29

2.3.3 Support Vector Machines (SVM) 31

2.3.4 Hidden Markov Models (HMM) 32

2.3.5 Related Work 33

2.4 Summary 46

3 Gesture Learning Module Architecture (GeLMA) **47**

3.1 Introduction 47

3.2 Pre-Processing and Hand Segmentation (PHS) module 51

3.2.1 Hand Detection and Tracking 51

3.2.2 Hand Segmentation 54

3.3 Static Gesture Interface (SGI) module 57
  3.3.1 Feature Extraction 57
  3.3.2 Train and Classification 60
3.4 Dynamic Gesture Interface (DGI) Module 64
  3.4.1 Feature Extraction 64
  3.4.2 Train and Classification 66
3.5 Vision-based Hand Gesture Recognition System Architecture 69
3.6 Summary 70

4 System Implementation 73
4.1 Introduction 73
4.2 Pre-processing and hand segmentation 73
4.3 Static Gesture Interface Module 75
  4.3.1 Feature extraction 77
  4.3.2 Support Vector Machine (SVM) Model Train 78
  4.3.3 Hand Posture Classification 79
4.4 Dynamic Gesture Interface Module 79
  4.4.1 Feature Extraction 81
  4.4.2 Learn HMM Parameters. 84
  4.4.3 Gesture Classification 84
4.5 Vision-based Hand Gesture Recognition System 84
4.6 Summary 87

5 Case Studies 89
5.1 Introduction 89
5.2 Vision-based Remote Hand Robot Control 90
  5.2.1 Introduction 90
  5.2.2 Hand Segmentation 91
  5.2.3 Orientation 94
  5.2.4 Prototype implementation 96
5.3 Vision-based Hand Robotic Wheelchair Control 98
  5.3.1 Introduction 98
  5.3.2 The Wheelchair Command Language Definition 100
  5.3.3 Hand Segmentation 101
  5.3.4 The k-curvature algorithm 102

5.3.5 Direction of turning 105
5.3.6 Prototype implementation 107
5.4 Referee CommLang Prototype 111
5.4.1 Introduction 111
5.4.2 The Referee Command Language Definition 111
5.4.3 Prototype Implementation 113
5.5 Sign Language Recognition Prototype 117
5.5.1 Introduction 117
5.5.2 Prototype Implementation 118
5.6 Summary 120

6 Experiments and Results 125
6.1 Introduction 125
6.2 Comparative study of machine learning algorithms for hand posture classification 126
6.2.1 Experimental Setup 126
6.2.2 Results 135
6.2.3 Discussion 138
6.3 Comparative study of seven different feature extraction algorithms 139
6.3.1 Experimental Setup 139
6.3.2 Results 154
6.3.3 Discussion 156
6.4 Integration of Static and Dynamic Gestures 158
6.4.1 Experimental Setup 158
6.4.2 Results 161
6.4.3 Discussion 165
6.5 A Comparative Study of Hand Features for Sign Language Recognition 167
6.5.1 Experimental Setup 167
6.5.2 Results 169
6.5.3 Discussion 170
6.6 Summary 171

7 Conclusions and Future Work 173
7.1 Synthesis of Developed Work 173
7.2 Main Results and Conclusions 174

7.3 Limitations ... 176
7.4 Major Contributions ... 176
7.5 Development Perspectives and Future Work ... 177

References ... 181

List of Figures

Figure 1. Example of a $LBP_{8,1}$ computation. 15
Figure 2. Artificial Neural Network [78] 30
Figure 3. Neural network with inputs, activation function and output represented. [79] 31
Figure 4. SVM: support vectors representation with maximum-margin hyperplane [84] 32
Figure 5. HMM example with initial probabilities (Π), transition probabilities (aij), observation probabilities (bjm) and observation sequence (O) represented. 33
Figure 6. The static and dynamic gesture training and learning architecture. 50
Figure 7. Pre-processing and feature extraction diagram. 51
Figure 8. Hand tracking in real-time (hand path represented in white). 52
Figure 9. Inside Kinect controller [134] 52
Figure 10. Kinect Interface [135] 53
Figure 11. Kinect RGB image + Depth image. 53
Figure 12. Bounding box size relative to the hand distance to the camera. 54
Figure 13. Hand bounding box with the near and far-threshold planes represented (left); 3D bounding box obtained for hand segmentation (right). 55
Figure 14. Hand blob extracted from the depth image and used for feature extraction. 55
Figure 15. Hand contour extracted from hand blob. 59
Figure 16. Centroid distance feature calculation diagram. 59
Figure 17. Centroid distance signature. 60
Figure 18. SVM train diagram. 61
Figure 19. Hand posture classification diagram, using trained SVM model. 62
Figure 20. “Close” command detected and correctly classified. 62
Figure 21. “Five” command detected and correctly classified. 62
Figure 22. User interface for the static gesture training and testing. 63
Figure 23. Dynamic gesture feature extraction diagram. 65
Figure 24. Gesture path with respective feature vector after labelling. 65
Figure 25. Learn HMM model parameters from a training set of observations. 66
Figure 26. Output the model with the highest score based on the recognized gesture. 67
Figure 27. "*GOAL*" command correctly identified. 67
Figure 28. "*DROP-BALL*" command correctly identified. 67
Figure 29. A 4-state left-right HMM model. 68

Figure 30. Dynamic gestures learning interface. 68
Figure 31. Vision-based hand gesture recognition system architecture. 70
Figure 32. The Referee CommLang finite state machine diagram. 85
Figure 33. Vision-based Remote Robot Control diagram. 91
Figure 34. Robot direction of movement relative to hand position. 93
Figure 35. Vectors used for robot heading calculation. 94
Figure 36. Robot heading dependent on hand rotation. 94
Figure 37. Computer connected to a Kinect camera for remote hand gesture robot control. 96
Figure 38. Human computer interface for the Vision-based Remote Hand Robot Control. 98
Figure 39. Vision-based Hand Robotic Wheel Chair Control diagram. 99
Figure 40. Finger commands used to drive the wheelchair. 100
Figure 41. Hand peak and valley point detection 102
Figure 42. Dot product between vectors A and B 102
Figure 43. Cross product between two vectors 103
Figure 44. Vectors used in the calculation of finger orientation 106
Figure 45. HCI for the Vision-Based Hand Robotic Wheelchair Control prototype 110
Figure 46. Referee CommLang Interface diagram with all the modules 114
Figure 47. The set of dynamic gestures defined and used in the *Referee CommLang*. 114
Figure 48. The set of hand postures trained and used in the *Referee CommLang*. 114
Figure 49. The "GOAL" gesture recognized. 115
Figure 50. The "GOAL, TEAM1" sequence recognized. 115
Figure 51. The "GOAL, TEAM1, PLAYER2" sequence recognized. 115
Figure 52. The "SUBSTITUTION" gesture recognized. 116
Figure 53. The "SUBSTITUTION TEAM-1" sequence recognized. 116
Figure 54. The "SUBSTITUTION TEAM-1 PLAYER-IN-1" sequence recognized 116
Figure 55. The "SUBSTITUTION TEAM-1 PLAYER-IN-1 PLAYER-OUT-3" sequence recognized 117
Figure 56. Sign Language Recognition Prototype diagram 118
Figure 57. Portuguese Language manual alphabet 118
Figure 58. The vowel A correctly classified 119
Figure 59. The vowel E correctly classified 119
Figure 60. The vowel I correctly classified 119

Figure 61. The vowel O correctly classified........ 120

Figure 62. The vowel U correctly classified........ 120

Figure 63. Hand *convexity defects* [28]. 127

Figure 64. Application user interface used for hand feature extraction. 128

Figure 65. First RapidMiner setup screen for the k-NN dataset 1 analysis. 130

Figure 66. Second RapidMiner setup screen for the k-NN dataset 1 analysis (training and testing). 131

Figure 67. First RapidMiner setup screen for the ANN dataset 1 analysis. 132

Figure 68. RapidMiner ANN DataSet1 analysis (second screen). 133

Figure 69. RapidMiner parameter optimization setup for the SVM dataset 1 analysis........ 134

Figure 70. Second RapidMiner screen setup for the SVM dataset 1 analysis. 134

Figure 71. HoG (Histogram of Gradients) operator feature extraction. 141

Figure 72. Local Binary Pattern operator feature extraction. 141

Figure 73. The Shi-Tomasi corner detector operator feature extraction 142

Figure 74. Radial Signature operator feature extraction........ 142

Figure 75. Radial Signature Fourier Descriptors (RSFD) operator feature extraction. 143

Figure 76. Centroid Distance operator feature extraction. 143

Figure 77. Centroid Distance Fourier Descriptors operator feature extraction. 144

Figure 78. The defined gesture vocabulary (1. palm, 2. fist, 3. one-finger, 4. two-fingers, 5. three-fingers, 6. four-fingers, 7. five-fingers, 8. ok, 9. move-left and 10. circle) 145

Figure 79. Main user interface for the Kinect video recording application........ 145

Figure 80. Posture recording. "**Fist**" and "**One Finger**" postures. 146

Figure 81. Posture recording pause period. 146

Figure 82. Main user interface for the videos processing application. 147

Figure 83. User data folder selection for feature extraction. 148

Figure 84. Operator type selection (in the image the Radial Histogram was selected)........ 148

Figure 85. RapidMiner configuration for ANN Radial Signature feature analysis with parameter optimization. 149

Figure 86. RapidMiner configuration for the k-NN Radial Signature Fourier Descriptors feature analysis with parameter optimization (first screen). 150

Figure 87. Validation configuration with an option for Log display........ 151

Figure 88. Example of Log table obtained during parameter optimization........ 151

Figure 89. The training and testing configuration window for the k-NN learner................ 152
Figure 90. RapidMiner configuration for SVM histogram of gradients feature analysis with parameter optimization (first screen).. 152
Figure 91. RapidMiner SVM cross validation and log parameter configuration window.. 153
Figure 92. SVM model training and Testing configuration.. 153
Figure 93. Static gesture feature extraction and model learning user interface.................... 159
Figure 94. Dynamic gestures feature extraction and model training user interface. 160
Figure 95. First Rapid miner setup screen for the Find Best Learner Process (Read dataset and Parameter Optimization)... 162
Figure 96. Second Rapid miner setup screen for the Find Best Learner Process (Validation).. 162
Figure 97. Third Rapid miner setup screen for the Find Best Learner Process (Select Sub-Process). 163
Figure 98. Fourth Rapid miner setup screen for the Find Best Learner Process (Learn algorithms definitions). 163
Figure 99. Centroid distance dataset SVM parameter optimization.................................... 164
Figure 100. Hand depth image.. 168
Figure 101. RapidMiner sign language dataset, kernel type and C value, parameter optimization 169

List of Tables

Table 1: Equivalence between motions in the image and transform domains 18

Table 2. Hand centroid distance feature vectors prior to normalization................................ 60

Table 3. The Referee CommLang state transition table. 85

Table 4. Command set definition with associated odes and text description 123

Table 5. Sample values for the 1st dataset after vector normalization. 127

Table 6. Sample values for the 2nd dataset after normalization. 128

Table 7. Artificial Neural Network (ANN) parameter setup used....................................... 131

Table 8. Parameters obtained after the optimization process for the SVM 133

Table 9. Accuracy obtained on both datasets with each algorithm, and time spent in the tests. 135

Table 10. k-NN confusion matrix for dataset 1 136

Table 11. k-NN confusion matrix for dataset 2 136

Table 12. Naïve Bayes confusion matrix for dataset 1 136

Table 13. Naïve Bayes confusion matrix for dataset 2 136

Table 14. ANN confusion matrix for dataset 1 137

Table 15. ANN confusion matrix for dataset 2 137

Table 16. SVM confusion matrix for dataset 1 137

Table 17. SVM confusion matrix for dataset 2 137

Table 18. Best learning algorithm, in terms of accuracy, for each of the datasets and parameters obtained during optimization. 154

Table 19. Radial signature dataset confusion matrix. 155

Table 20. Centroid distance dataset confusion matrix. 155

Table 21. Radial signature Fourier confusion matrix. 155

Table 22. Centroid distance Fourier confusion matrix. 155

Table 23. Local binary patterns dataset confusion matrix. 156

Table 24. Histogram of gradients dataset confusion matrix. 156

Table 25. Shi-Tomasi corner detector confusion matrix. 156

Table 26. Centroid distance confusion matrix 165

Table 27. Hidden Markov Models accuracy for each gesture defined (Figure 47) 165

Table 28. Obtained kernel type and C value for the two datasets through parameter optimization 169

Table 29. Obtained accuracy for each dataset. 170

Table 30. Centroid distance features confusion matrix 170

Table 31. Distance features confusion matrix.. 170

List of Algorithms

Algorithm 1. Histogram of gradients computation (HoG) 13
Algorithm 2. Local Binary Pattern Image Computation 16
Algorithm 3. Compute Centroid Distance Signature Fourier Descriptors 18
Algorithm 4. k-NN classification algorithm... 29
Algorithm 5. Hand Segmentation... 73
Algorithm 6. Hand Inside Computation 74
Algorithm 7. Get Hand Image 75
Algorithm 8. Hand posture learn and classification algorithm... 76
Algorithm 9. Centroid Distance Computation.. 77
Algorithm 10. Train Static Gesture Classifier 78
Algorithm 11. Predict Hand Posture Class.. 79
Algorithm 12. Dynamic gesture learn and classification algorithm...................................... 80
Algorithm 13. Add gesture points to Trainset 81
Algorithm 14. Label sample points according to defined alphabet...................................... 81
Algorithm 15. Rescale points to follow inside predefined hand viewport 82
Algorithm 16. Label point according to defined alphabet.. 83
Algorithm 17. Gesture classification for a given hand path... 84
Algorithm 18. Vision-based hand gesture recognition system... 86
Algorithm 19. Nearest point computation 92
Algorithm 20. Angle and hand orientation computation.. 95
Algorithm 21. *Vision-based Remote Robot Control*.. 97
Algorithm 22. Finger tip detection and extraction 103
Algorithm 23. Finger point computation... 105
Algorithm 24. Orientation computation based on finger tips... 106
Algorithm 25. Vision-based Hand Robotic Wheel Chair Control 108

Chapter 1

1 Introduction

1.1 Motivation

Hand gesture recognition for human computer interaction is an area of active research in computer vision and machine learning. One of the primary goal of gesture recognition research, is to create systems, which can identify specific gestures and use them to convey information or to control a device. Though, gestures need to be modelled in the spatial and temporal domains, where a hand posture is the static structure of the hand and a gesture is the dynamic movement of the hand.

Being hand-pose one of the most important communication tools in human's daily life, and with the continuous advances of image and video processing techniques, research on human-machine interaction through gesture recognition led to the use of such technology in a very broad range of applications, like touch screens, video game consoles, virtual reality, medical applications, among others.

There are areas where this trend is an asset, as for example in the application of these technologies on interfaces that can help people with physical disabilities, or areas where it is a complement to the normal way of communicating.

There are basically two types of approaches for hand gesture recognition: vision-based approaches and data glove methods.

This work focus on creating a vision-based approach, to implement a system capable of performing gesture recognition for real-time applications. Vision-based hand gesture recognition systems were the main focus of the work since they provide a simpler and more intuitive way of communication between a human and a computer. Using visual input in this context makes it possible to communicate remotely with computerized equipment, without the need for physical contact.

In the context of these research areas, it is important to mention the RoboCup competition, a challenging international research and educational initiative, being held every year since 1997, that provides a test-bed where a significant number of technologies can be experienced and integrated [1, 2]. Every year, new technical challenges are presented, and the progress in fields like intelligent robotics, artificial intelligence (AI) and applied technology are extremely relevant, especially in the Middle Size League (MSL) [1], and RoboCup@Home league. The RoboCup@Home league aims to develop service and assistive robot technology with high relevance for future personal domestic applications [3]. On the other hand, the RoboCup MSL games, use real wheeled robot teams, to play with an ordinary soccer ball, autonomously. One referee and at least one assistant are assigned for judgment of a match. They use assisting technology, the RefereeBox, to support them, in particular for conveying referee decisions for players, with the help of a wireless communication system.

Also, the possibility to have systems able to interpret sign language in real-time is an important aspect to take into account. Those systems could be used to facilitate the communication between humans and machines and help disabled people or people with physical limitations taking care of generic domestic tasks.

As previously said, the main objective of this work consists of studying and implementing solutions, generic enough, with the help of machine learning algorithms, allowing their application in a wide range of human-computer interfaces, for online gesture recognition. In pursuit of this, it is intended to use a depth camera to detect and track the user hands, and extract information (hand features), for gesture classification. With the implemented solutions it is intended to develop an integrated vision-based hand gesture recognition system, for offline training of static and dynamic hand gestures, in order to create models, that can be used for online classification of user commands, that could be defined with the help of a new formal language.

The motivation to apply the developed technologies in the present study, in a vision-based system for hand gesture recognition relates to a number of factors here highlighted, such as the interest in creating:

- Flexible systems in terms of configuration and capacity of being integrated with any human-computer interface.
- Robust user-independent gesture recognition systems.
- Systems that can easily learn new gestures that could be used in any human-computer interface.

- Systems that can easily integrate static and dynamic gesture recognition.
- Systems that can be used in any type of environment.
- Systems that could easily improve the identification of commands and communication between referees in noisy sports and gaming environments.
- Systems with improved response time, since the command identified by the vision system is automatically transmitted to the human-computer interface (avoiding delays due to misinterpretations).
- Real-time sign language recognition systems.
- Real-time robotic soccer refereeing systems.
- Generic human-machine interaction systems that may control any type of electronic device with configurable static and dynamic gestures.

Those solutions could be used to build a wide range of human-computer interfaces, as for example, a system able to remotely drive a robot, or in a vision-based wheelchair control, a sign language recognition system or on any other kind of robot/system command interface that can take advantage of them.

1.2 Objectives

The main objective of this work is to build an integrated vision-based system able to interpret a set of defined commands composed of static and dynamic gestures. That system should provide the ability to quickly learn new gestures and be configured to recognize new commands. The solutions should be generic and easily applied to a wide range of applications where the core of vision-based interaction is the same thereby facilitating its implementation.

For that, four specific objectives must be achieved by the research, answering the following questions:

- Is it possible to build a set of solutions that can be easily configured to learn different static or dynamic gestures for online classification, and be at the same time easy to extend and easy to be used by future users?
- Is it possible to define a new formal command language, capable of representing all the system commands or combinations of all the static and

dynamic gestures defined, being at the same time simple in its syntax and easy to use in any system configuration?

- Are the implemented solutions generic enough so that it is possible to use them to build any vision-based application for a robot/system command interface?
- Is it possible to build a set of solutions, generic enough, able to interpret user gestures and apply them to the problem of robotic soccer referee’s commands classification, in order to help him judge a game in real time?

1.3 Approach

In order to try to answer the questions outlined in the objectives the following approaches will be pursued:

- Identification of hand features that best fit the problem at hand, being at the same time simple in terms of computational complexity, for use in real-time applications.
- Test the selected features with machine learning algorithms in order to identify the best learning algorithm in terms of classification.
- Build solutions, that can be easily configured to recognize a specific set of static hand postures, thus allowing to learn a model that could be used in any online classification system.
- Build solutions, that can be easily configured to recognize a specific set of dynamic gestures, thus allowing to learn models that could be used in any online classification system.
- Build a system that integrates the two types of gestures, and test it in real-time situations.
- Train and test the system so that it is able to interpret the set of commands, defined with the new formal language.

1.4 Contributions

This work proposes a new generic vision-based hand gesture recognition architecture, able to be integrated with any specific human computer interaction (HCI) system.

The description of this scientific contribution is split into two groups. The first group is based on the initial work carried out during the hand detection, tracking and segmentation problems. The various experiments carried out during this phase allowed to conclude that it would be more advantageous to use a camera that could output two images, a colour image and a distance image. Therefore, the best option was to use the Kinect camera [4] which measures the time-of-flight of a light signal between the camera and the objects of interest, for each point of the image.

Some prototypes were therefore developed, which allowed to reach the first group of contributions, here listed:

- Definition of a formal command language, used in a vision-based system prototype, able to interpret a number of finger commands that are used to drive a robotic based wheel chair [5]. The language was defined using BNF (Bakus Naur form).
- Implementation of a system able to remotely drive a robot with a finite set of hand gestures [6], and that can be easily adapted to any system for remote robot operation.

The second group consists of solving the four specific objectives defined in the first section, and they include the following contributions:

- The implementation of two applications able to train and learn statistical based models, for any static or dynamic gesture definition, that may be integrated in any system for real time hand gesture classification.
- The implementation of a generic system able to use static, dynamic or a combination of both to classify any gesture in real-time, using the models learned in the train phase [7].
- The integration of machine learning algorithms to increase the performance and effectiveness of real time static and dynamic gesture classification [8].

- A new language definition, the *Referee CommLang*, which is a formal definition of all the commands that the implemented gesture recognition system accepts. The language was defined with BNF (Bakus Naur form) [9].

1.5 Document Structure

The reminder of this document is divided into seven chapters and is organized as follows:

- Chapter 1 introduces the motivation and objectives of this work. The four specific objectives to be achieved by this work are also discussed together with the intended approach.
- Chapter 2 presents and discusses the foundations necessary to understand the scientific and technical concepts involved in the study. The state of the art in the areas of hand feature extraction and detection, data-mining techniques, static and dynamic gesture recognition, and the use machine-learning techniques for pattern classification is also presented.
- Chapter 3 describes in detail the three implemented modules: the Pre-Processing and Hand Segmentation module, the Static Gesture Interface and the Dynamic Gesture Interface. The modules allow the training of all the gestures that will be part of the system, and learn the models that can be used in any interactive system for vision-based hand gesture recognition. This chapter also discusses all the assumptions that the system must obey.
- Chapter 4 presents and describes the algorithms implemented and used to build all the models from the previous chapter.
- Chapter 5 presents some case studies implemented with solutions developed during the study, which resulted in prototypes with potential for being integrated into other systems.
- Chapter 6 presents the experiments performed in terms of hand feature extraction, selection and model learning, for static and dynamic hand gesture recognition. It discusses the tools used in order to obtain the optimized parameters for the learning algorithms and the process of final model

selection. At the end of each section, the obtained results are analysed and discussed.

- The document ends with a chapter that gives an overview of the work reported in this thesis, underlining the tools and results achieved and provides some future directions.

Chapter 2

2 Methodologies and State of the Art

2.1 Introduction

Vision-based hand gestures interaction is a challenging interdisciplinary research area, which involves areas such as computer vision and graphics, image processing, data mining, machine learning, and informatics. To make use of such techniques in a successful working system, there are some requirements which the system should satisfy [10] such as for example robustness to variations in illumination and to occlusions. The system must be computationally efficient, so it can be used in real-time situations, and it must be error tolerant, giving users the possibility to repeat some actions. Also, it should be flexible enough so that it can be adapted to different scales of applications, i.e., the core of vision-based interaction should be the same for desktop environments and for robot control, for example.

Vision based gesture recognition has the potential to be a natural and powerful tool supporting efficient and intuitive interaction between humans and computers. Visual interpretation of hand gestures can help achieving the ease and naturalness desired for HCI (Human Computer Interaction). Also, vision has the potential of carrying a wealth of information in a non-intrusive manner at a low cost. Therefore, it constitutes a very attractive sensing modality for developing hand gesture recognition, and can be divided into two categories: *3D model based methods* and *appearance based methods* [11, 12].

3D model-based methods are used to recover the exact 3D hand pose. Such models however have a disadvantage which is computationally intensive, making such methods less suitable for real-time applications. On the other side, although appearance-based methods are view-dependent, they are more efficient in terms of computation time. They aim at recognizing a gesture among a vocabulary, with template gestures learned from training data [10, 13, 14].

Appearance-based models extract features that are used to represent the object under study and must have, in the majority of cases, invariance properties to translation, rotation and scale changes.

Given this, the approach typically used for vision-based hand gesture recognition interfaces can be divided into the following three steps: ***hand detection and tracking***, ***hand feature extraction*** and ***hand gesture classification***.

There are many studies on gesture recognition and methodologies well presented in the literature [10, 13-16].

The following sections present some background and the state of the art related with the areas of hand feature selection and extraction, static or hand posture classification and dynamic gesture classification. The importance of proper feature selection and extraction will be addressed and a comprehensive description of some of the techniques and algorithms used in the area will be given.

2.2 Background

2.2.1 Feature Selection and Extraction

For vision-based hand gesture recognition and classification, careful feature selection for hand shape representation plays an important role. Since visual features provide a description of the image content [17], their proper choice for image classification is vital for the future performance of the recognition system. Viewpoint invariance and user independence are two important requirements for a real-time hand gesture recognition system.

Thus, efficient shape features must present some essential properties such as [17]:

- **Translation, rotation, and scale invariance**: meaning that the location, rotation and scaling changing of the shape must not affect the quality and robustness of extracted features.
- **Occlusion invariance**: when some parts of a shape are occluded by other objects, the features of the remaining part must not change compared to the original shape.
- **Noise resistance**: features must be as robust as possible to noise, or the shape must be classified the same way independently of the noise strength present.
- **Reliable**: as long as one deals with the same pattern, the extracted features must remain the same.

A shape descriptor, normally in the form of a vector, is a set of numbers that are produced to describe the object that the system needs to identify. Descriptors attempt to quantify shapes in ways that agree with human intuition, and should meet the following requirements:

- They should be as complete as possible.
- They should be represented and stored compactly, namely the size of the vector must not be too long.
- The computation of distance between descriptors should be simple in order to optimize processing times.

Many shape descriptors and similarity measures have been used to date in the field of gesture recognition for human-computer interaction. A good survey on shape feature extraction techniques is presented by Mingqiang [17] and Bourennane [18].

The following sections will describe and give the corresponding algorithm implementations for a set of image features tested throughout the study, namely the Centroid Distance Signature [19], the Histogram of Oriented Gradients (HoG) [20-22], the Local Binary Pattern (LBP) [23], the Fourier Descriptors (FD) [12, 18, 24, 25] and the Shi-Tomasi Corner detector [26-29].

2.2.2 Centroid Distance Signature

The centroid distance signature is a type of shape signature. This signature is expressed by the distance of the hand contour boundary points, from the centroid (x_c, y_c) of the shape and given by the following formula:

$$d(i) = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2}; i = 0, \dots, N \tag{1}$$

where $d(i)$ is the calculated distance, and (x_i, y_i) are the coordinates of contour points.

Due to the subtraction of centroid, which represents the hand position, from boundary coordinates, the centroid distance representation is invariant to translation. Rayi Yanu Tara et al. [19] demonstrated this property and also, that a rotation of the hand by an amount θ results in a circularly shift version of the original image. Thus, this shift version can be interpreted as a different gesture, or rotation invariance can be achieved by shifting the obtained signature by the rotation amount. The centroid distance signature algorithm implementation is given and discussed in section 4.3.1, since this was the type of feature selected for the final system implementation.

2.2.3 Histogram of Oriented Gradients (HoG)

The essential thought behind the Histogram of Oriented Gradient descriptors is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. Since the HoG descriptor operates on localized cells, the method is invariant to geometric transformations and illumination changes.

Pixel intensities can be sensitive to lighting variations, which can lead to classification problems within the same gesture under different light conditions. So, the use of local orientation measures avoids this kind of problem, and the histogram gives us translation invariance. Orientation histograms summarize how much of each shape is oriented in each possible direction, independent of the position of the hand inside the image [30]. This statistical technique is most appropriate for close-ups of the hand.

Although the method is insensitive to small changes in the size of the hand, it is sensitive to changes in hand orientation. Again, as in the previous descriptor, this can be used to identify a different gesture, or by histogram shifting by the rotation amount we can get orientation invariance. The local orientation is calculated using image gradients, represented by horizontal and vertical image pixel differences. If d_x and d_y are the outputs of the derivative operators, then the gradient direction is given by $atan2(d_y, d_x)$, and the contrast by $\sqrt{d_x^2 + d_y^2}$.

After histogram calculation, a blur in the angular domain is applied, which allows a gradual fall-off in the distance between orientation histograms as explained by William T. Freeman et al. [31]. Similar to the implementation described in the paper, we used the same kernel as defined in the algorithm on line 40.

The implementation of the histogram of gradients and the respective histogram blur computation is shown in the following algorithm.

Algorithm 1. Histogram of gradients computation (HoG)

```
inputs:
  image :  a matrix containing the grey-level hand pixels

outputs:
  histogram : a vector containing the histogram of gradient values

begin
  histSize ← 36
  initializeHistogramToZero()
  rows ← getNumbersOfRows(image)
  cols ← getNumberOfCols(image)

  gradientThreshold ← 1.2f
  dx ← d(image) / dx
  dy ← d(image) / dy
  contrast ← sqrt($dx^2 + dy^2$)

  sum ← 0
  for  i = 0 to rows - 1 do
    for j = 0 to cols - 1 do
      sum ← sum + contrast[i, j]
    endfor
  endfor

  grayAverage ← sum / (rows * cols)
  step ← 360 / histSize

  //----- define a threshold to eliminate low contrast gradients -----
  threshold ← grayAverage * gradientThreshold
  for i = 0 to rows - 1 do
    for j = 0 to cols - 1 do
      if contrast[i, j] > threshold then
        histogram[contrast[i, j] / step] ← histogram[contrast[i, j] / step] + 1
      endif
    endfor
  endfor

  //------------------- blur the histogram -------------------
  temp[histSize] ← 0
  filter ←{1, 4, 6, 4, 1}
  total ← $\Sigma_i$ filter[i]

  for i = 0 to histSize - 1 do
    sum ← 0
    for j = 0 to size(filter) - 1 do
      if (i-2+j) ≥ 0 and (i-2+j) < histSize then
        sum ← sum + (histogram [(i-2+j)] * filter[j])
      elseif (i-2+j) < 0  then
```

```
49.             sum ← sum + (histogram [histSize + (i-2+j)] * filter[j])
50.         elseif ((i-2+j) ≥ histSize) then
51.             sum ← sum + (histogram [ (i-2+j) % histSize] * filter[j])
52.         endif
53.       endfor
54.       temp[i] ← sum / total
55.     endfor
56.
57.     for i = 0 to histSize - 1 do
58.       histogram[i] ← temp[i]
59.     endfor
60.
61.     return histogram
62.   end
```

The algorithm first calculates the image gradients on line 14 and 15 using one of many possible operators. In the proposed solution implement the Sobel operator was used [32-34]. This operator is used to find the approximate absolute gradient magnitude at each point in an input grey scale image. The grey average from the obtained contrast image is used to define a contrast threshold used to eliminate low gradient values during histogram calculation. Also, the number of histogram bins, or orientation values, was defined to be 36, so, the contrast image values are mapped to the proper histogram range by dividing by the step value defined in line 26. After histogram calculation, the respective histogram blur is performed with the defined filter initialized in line 40.

2.2.4 Local Binary Patterns

Local binary patterns (LBP) is a grey scale invariant local texture operator with powerful discrimination and low computational complexity [23, 35-37]. This operator labels the pixels of the image by *thresholding* the neighbourhood of each pixel $g_{p\ (p=0\dots P-1)}$, being P the values of equally spaced pixels on a circle of radius R $(R > 0)$, by the grey value of its centre g_c and considers the result as a binary code that describes the local texture [23, 35, 37]. The binary code is calculated according to the following formula:

$$LBP_{P,R} = \sum\nolimits_{(p=0,\dots,P-1)} s\left(g_p - g_c\right) 2^p \tag{2}$$

where,

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3}$$

Figure 1 illustrates the computation of $LBP_{8,1}$ for a single pixel in a rectangular 3x3 neighbourhood with the threshold version in the middle and the resulting binary code on the right. The pixel g_0 is always assigned to be the grey value of the neighbour to the right of g_c.

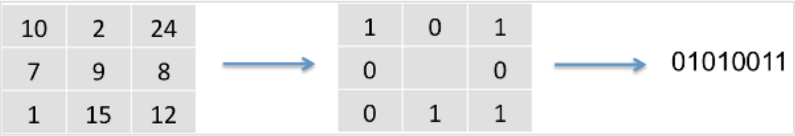


Figure 1. Example of a $LBP_{8,1}$ computation.

In the general definition, LBP is defined in a circular symmetric neighbourhood, which requires interpolation of the intensity values for exact computation. The coordinates of g_p are given by (-R·*sin*(2πp/P), R·*cos*(2πp/P)) [35].

The $LBP_{P,R}$ operator produces 2^P different output values, corresponding to the different binary patterns that can be formed by the P pixels in the neighbourhood set.

As a rotation of a textured input image causes the LBP patterns to translate into a different location and to rotate about their origin, if rotation invariance is needed, it can be achieved by rotation invariance mapping. In this mapping, each LBP binary code is circularly rotated into its minimum value using the following formula:

$$LBP_{P,R}^{ri} = min_i\left(ROR\left(LBP_{P,R}i\right)\right) \tag{4}$$

where *ROR*(*x i*) denotes the circular bitwise right shift on the P-bit number x, i steps. For example, 8-bit *LBP* codes 00111100b, 11110000b, and 00001111b all map to the minimum code 00001111b. For P=8 a total of 36 unique different values is achieved. This operator was designated as *LBPROT* in [38].

Ojala et.al [35] had shown however, that *LBPROT* as such does not provide very good discrimination. They have observed that certain local binary patterns are fundamental properties of texture, providing the vast majority of all 3x3 patterns presented in observed textures. They called this fundamental patterns "*uniform*" as they have one thing in common – uniform circular structure that contains very few spatial transitions. They introduced a uniformity measure *U(pattern)*, which corresponds to the number of spatial transitions (bitwise 0/1 changes) in the "*pattern*". Patterns that have a U value of at most 2 are designated uniform and the following operator for grey-scale and rotation invariant texture description was proposed:

$$LBP_{P,R}^{riu2} = \begin{cases} \sum_{(p=0,\ldots,P-1)} S(g_p - g_c), & if\ U(LBP_{P,R}) \leq 2 \\ (P+1), otherwise \end{cases} \tag{5}$$

Equation 5 assigns a unique label corresponding to the number of '1' bits in the uniform pattern, while the non-uniform are grouped under the "*miscellaneous*" label *(P+1)*. In practice the mapping from $LBP_{P,R}$ to $LBP_{P,R}^{riu2}$ is best implemented with a lookup table of 2p elements. The final texture feature employed in texture analysis is the histogram of the operator output.

The local binary pattern implementation is given by the following algorithm.

Algorithm 2. Local Binary Pattern Image Computation

```
inputs:
  image :  a matrix containing the binary-segmented hand pixels
  neighbours: the number of neighbours (default = 8)
  radius: the radius value (default = 1)
outputs:
  lbpImage :  a matrix containing the final calculated LBP's

begin
  neighbours ← max(min(neighbours, 31), 1)
  rows ← getNumberOfRows(image)
  cols ← getNumberOfCols(image)

  for  n = 0 to neighbours - 1 do
    x ← (radius) (cos(2πn / (neighbours)))
    y ← (radius) (-sin(2πn / (neighbours)))
  endfor

  fx ← floor(x)
  fy ← floor(y)
  cx ← ceil(x)
  cy ← ceil(y)
  ty ← y - fy
  tx ← x - fx

```

```
    w1 ← (1 - tx) * (1 - ty)
    w2 ← tx * (1 - ty)
    w3 ← (1 - tx) * ty
    w4 ← tx * ty

    for i = radius to (rows – radius) - 1 do
      for j = radius to (cols – radius) - 1 do
        t ← w1 * image [i + fy, j + fx] + w2 * image [i + fy, j + cx)]
            + w3 * image [i + cy, j + fx] + w4 * image [i + cy, j + cx]
      endfor
    endfor

    lbpImage[i - radius, j - radius] ← lbpImage[i - radius, j - radius]
                                    + ((t > image[i, j]) ^ (abs(t - image[i, j]))
  end
```

2.2.5 Fourier Descriptors

Instead of using the original image representation in the spatial domain, feature values can also be derived after applying a Fourier transformation. The feature vector obtained this way is called a Fourier descriptor [39]. This is another feature normally used to describe a region boundary [25, 33], and considered to be more robust with respect to noise and minor boundary modifications.

For computational efficiency, the number of points is chosen to be a power of two [12]. The normalized length is generally chosen to be equal to the calculated histogram signature length (N). Hence the Fourier Transform we obtain N Fourier coefficients C_k given by the following formula:

$$C_k = \sum_{i=0}^{N-1} z_i e^{\left(\frac{2\pi jik}{N}\right)}, \; k = 0, \dots, N-1 \tag{6}$$

Table 1 shows the relation between motion in the image and the transform domains, which can be used in some types of invariance.

The first coefficient C_0 is discarded since it represents, in our case, the hand position. Hand rotation affects only the phase information, thus if rotation invariance is necessary, it can be achieved by taking the magnitude of the coefficients.

Table 1: Equivalence between motions in the image and transform domains

In the image	In the transform
A change in size	Multiplication by a constant
A rotation of Ø about the origin	Phase shift
A translation	A change in the DC term

Division of the coefficients by the magnitude of the second coefficient, C_1, on the other hand, achieves scale invariance. This way, N-1 Fourier descriptors I_k are obtained:

$$I_k = \frac{|C_k|}{|C_1|}, k = 2, \dots, N-1 \tag{7}$$

Conceil et.al [12], showed that with 20 coefficients the hand shape is well reconstructed. Centroid distance Fourier descriptors, obtained by applying the Fourier transform on a centroid distance signature, were empirically proven to have higher performance than other Fourier descriptors [19, 25, 40].

The following algorithm implements the centroid distance Fourier descriptors with the help of the FFTW library [41]. The algorithm receives a vector with the centroid distance signature and builds a complex vector with the real part equal to the centroid distance value and the imaginary part initialized to zero. It uses two library functions for the Fourier calculation: the *fftw_plan_dft_1d* function to build the pretended Discrete Fourier Transform plan, and the *fftw_execute* function to execute the plan and output the vector with the Fourier Descriptors.

Algorithm 3. Compute Centroid Distance Signature Fourier Descriptors

```
inputs:
  centroidDist: vector containing the centroid distance signature
outputs:
  out : fft_complex  vector variable that will contain the descriptors

begin
  N ← 20
  planForward ← fftw_plan_dft_1d(N, in, out,
                      FFTW_FORWARD, FFTW_ESTIMATE)
  for i = 0 to N – 1 do
    in[i][1] ← centroidDist[i]
    in[i][1] ← 0.0
  endfor
```

```

    $out \leftarrow$ fftw_execute(planForward)
end
```

2.2.6 The Shi-Tomasi Corner Detector

The Shi-Tomasi corner detector algorithm [29] is an improved version of the Harris corner detector [27]. The proposed improvement is in how a certain region within the image is scored and thus treated as a corner or not. Where the Harris corner detector determines the score R with the eigenvalues λ_1 and λ_2 of two regions in the following way:

$$R = det(\lambda_1 \lambda_2) - k(\lambda_1 + \lambda_2)^2 \tag{8}$$

where the second region is a shifted version of the first one, and if the difference between the two is big enough one can say it is a corner, Shi and Tomasi just use the minimum of both eigenvalues in the following way:

$$R = min(\lambda_1, \lambda_2) \tag{9}$$

and if R is greater than a certain predefined value, it can be marked as a corner. They demonstrated experimentally in their paper, that this score criterion is much better. Corner feature extraction information was implemented with the help of the OpenCV [28] library algorithms, so no algorithm implementation is presented here.

2.2.7 Related Work

Many authors have used the Histogram of Oriented Gradients (HoG) as a feature vector for gesture classification. As explained in section 2.2.3 it has some advantages over other methods and is simple and fast to compute. This technique is sometimes used in conjunction with others to improve the quality of features or solve some of the problems that one technique has per se.

Freeman et al. [31] presented a method for real-time hand gesture recognition based on this technique. They used the HoG as a feature vector for static gesture classification and interpolation. They implemented a real-time system able to distinguish a small vocabulary of about ten different gestures. Some problems arises form this approach namely: different gestures may have similar orientation histograms and the hand must dominate the image, i.e. the method is most appropriate for hand close-ups as explained in section 2.2.3.

Yafei Zhao et al. [21] have also presented a real-time hand gesture recognition method, where the histograms of oriented gradients (HoG) are used to describe the hand image. The HoG features are computed by dividing the image into overlapped blocks and these blocks are divided into smaller un-overlapped spatial regions, which are called cells. For each cell a local one-dimensional histogram of gradient directions is accumulated over the cell pixels. Local contrast normalization is required due to variations in illumination, so block-level histograms are normalized using L2-Hys scheme, which is a combination of normalization by the L2-norm and then limiting the maximum values to a predefined level and re-normalizing. The HoG feature vector is then the concatenated histogram entries from all the cells. The extracted HoG features are projected into a low-dimensional subspace using PCA-LDA (Principal Component Analysis and Linear Discriminant Analysis), since the dimensionality of the HoG feature is very high. This way, they were able to project a HoG feature from a 1296-dimensional space into a 9-dimensonal sub-space. After projecting all the training samples, the mean for each class is calculated and a classifier using the nearest neighbour method is constructed. Their system was able to recognize gestures in real-time however, the method still had some problems with some of the gestures, caused by shadows in the hand, leading to a detection rate lower than 80%.

Hanning Zhou et al. [42], Xingbao et al. [22] and Liang Sha et al. [43] proposed new methods for static hand gesture recognition based on an extended Histogram of Oriented Gradients features. Hanning Zhou et al. method augments the HoG feature with its relative image coordinates, and clusters the resulting vector in order to find a compact yet descriptive representation of the hand shape. To extract HoG features they used overlapping 24x24 sub-windows divided further into 4x4 blocks. Within each pixel block, they collected histograms using 8 bins ranging form 0 to π, obtaining a total of 128 feature elements in the local orientation histogram. The resulting vector is normalized to unit vector. They augmented the histogram with the image coordinates of the upper left corner of the sub-window, centralized to the median of the coordinates of all pixels belonging to the hand region to ensure robustness against in-plane translation. To obtain a compact representation they used the k-means clustering algorithm and recorded the mean vectors. Searching the nearest neighbour in the database of mean features, using a Euclidean distance metric, does hand posture recognition. The experiments showed that the purposed method was able to capture the distinct hand shape without requiring clean segmentation. Also, they were able to achieve higher recognition rates when compared to other techniques, and with a better efficiency in terms of computationally speed. On the other hand, Xingbao model uses a skin colour histogram of oriented gradients, to construct a human hand detector. Histogram of oriented Gradient features was first used by Navneet Dalal [44] with excellent results in human detection. Skin colour is the most commonly used cue for segmenting hands or faces in gesture analysis. However, approaches based on predefined skin colour models suffer from sensitivity to illumination variations. To solve this problem, the authors combined skin colour cues with HoG features to construct a novel feature: SCHOG. The SCHOG extracted features are used to train a SVM and construct a classifier able to detect the users hands. They tested the method on a SCHOG dataset and compared the obtained results on an unchanged HoG features dataset. The experimental results showed that the SCHOG features exhibited a good performance on the testing dataset with a detection rate of 97,8% compared to 94,4% on the normal HoG features. Liang Sha et al. presented a framework for hand posture recognition in consecutive video frames that used a Mixture of Gaussians to construct a model able to segment skin/non skin hand regions and used a particle filter [45] to track the hand. The model is trained and updated online to improve hand detection and tracking. To extract the Histogram of Oriented Gradients (HoG) features descriptor, the input region is divided into 4 cells on which the X and Y gradients are calculated. The gradient orientation and magnitude are calculated and accumulated in a local 1-D histogram. The calculated histograms for all the cells are joined and contrast-normalized to form the final HoG. The authors used the HoG descriptor not only as a recognition cue, but also for calibrating coarse tracking results. The extracted features are then used to select the class with the help of a posture dataset. They noticed that the gradient information was disturbed with cluttered backgrounds or with ill white balance. The system had some problems with over-exposure and drastic hand motion since the colour feature based tracker, under the above two conditions, could not separate correctly the hand from the background. Zondag et al. [20] tested the use of HoG features with two variations of the Adaboost algorithm to construct a real-time hand detector for indoor environments with cluttered backgrounds and variable illumination. The two variations of Adaboost, Gentle [46] and Discrete Adaboost [47], were tested with s*tump*, a machine-learning model consisting of a one level decision tree, and *Tree* weak classifiers. The HoG features were compared with Haar-like features and they found that the first ones presented a better performance. They recorded four different persons performing the "open hand" pose with different backgrounds and varied illumination conditions. From the acquired images they constructed three databases with the same samples but with different sizes. During the experiments, they used 2/3 of the dataset for training and 1/3 for testing and they performed four experiments: search of optimal HoG parameters, influence of database size, influence of the database example size and comparison of HoG and Haar-like features. The experiments showed that a real-time hand detector using HoG image features could achieve similar performances to a detector using Haar-like features on the created databases, although Haar-like features detectors could perform the detection roughly twice as fast. The HoG features on the other side have the advantage of having smaller feature vector, so it can be used in conjunction with much larger databases. Also, the HoG features detector consistently achieved better average false positive rates than Haar-like features detectors.

Mohamed Bécha Kaâniche et al. [48] introduced a new HoG tracker for gesture recognition based on local motion learning. They built HoG trajectory descriptors (representing local motion) by first selecting in the scene a set of corner points to determine textured regions, where they compute the 2D HoG descriptors. Then, they track these descriptors, with a new tracking algorithm based on a frame to frame HoG tracker and using an extended Kalman filter, in order to build temporal HoG descriptors. A newly computed descriptor initializes a new tracking process through the extended Kalman filter. The temporal 2D descriptor is the vector obtained by the concatenation of the final descriptor estimate and the positions of the descriptor during the tracking process. The learning / recognition of gestures is based on the k-means clustering algorithm and the k-nearest neighbour classifier. The extracted local motion descriptors are extracted from each video on the dataset annotated with the corresponding gesture and clustered into k (value defined empirically) clusters. With the obtained database of all the clusters, the k-nearest neighbour classifier is used to classify the gestures in the test dataset. They tested the tracking algorithm on two datasets: a synthetic dataset and a real dataset. They have validated the local motion descriptors with the KTH database [49]. They were able to achieve an accuracy of 91,33% when using the Shi-Tomasi corner detector [29] (section 2.2.6), and an accuracy of 94,67% when using the FAST (Features from Accelerated Segment Test) corner detector [50].

Local Binary Patterns (LBP) has also been used in some of the implementations described in the literature, often used in combination with other methods. As explained in section 0, it is a grey scale invariant local texture operator with powerful discrimination and low computational complexity. Marek Hrúz et al. [51] presented a description of appearance features using this technique, introduced by Ojala [52], to describe the manual component of Sign Language. They made experiments with these descriptors in order to show its discriminative power for hand shape classification. Then, they compared the performance of these features with geometric moments describing the trajectory and shape of hands. They tested the recognition performance of individual features and their combination on a dataset consisting of 11 signers and 23 signs with several repetitions. In their experiments, Local Binary Patterns outperform the geometric moments. LBP features showed better results than geometric moments for SSD (signer semi-dependent) and SI (signer independent) tests. When the features were combined they achieved a recognition rate up to 99,7% for signer dependent tests and 57,54% for signer independent tests. One drawback of the appearance-based features is their strong dependency on the position and orientation of the hand relative to the camera. Jinbing Gao et al. [53] proposed a new method which they called Adaptive HoG-LBP detector, to track the palm in unfettered colour images, by fusing HoG features and LBP features. They defend that the object description ability of the HoG and the LBP are different, so first, they determine the strength of each descriptor on the hand image or part of the image based on what they call a Confidence Map Computing. The extracted features are then fused based on the calculated confidence values and are normalized. The resulting features were then used to train an SVM classifier. They built their own dataset with 5378 palm images from 28 different users, and were able to achieve an accuracy of 95,2% in real-time situations. The classification is done without any pre-processing such as noise filtering, brightness balance or sharpening. The system showed however some problems with hand motion detection, which was defined as future work.

Instead of using the original image representation in the spatial domain, many authors propose hand feature representation based on Fourier Descriptors (FD) that, as explained in section 2.2.5, being another feature describing the boundary of a region are considered to be more robust with respect to noise and minor boundary modifications.

Conseil et al. [12], Barczak et al. [54] and Rayi et al. [19] proposed vision based systems for hand posture recognition based on this technique. Conceil et al. discussed the invariance properties of FD, and they provided a comparison of performances with Hu moments [55]. They made experiments with the Triesch hand posture database [56], and with their own database, constructed from images acquired from 18 non-expert users, performing the hand postures defined in their own gesture vocabulary. The database was build from a very large set of images, with various postures of the hand in order to test extensively the performances of FD in regard to invariances. Their work showed that Fourier descriptors were able to give good recognition rates in comparison with Hu moments, confirming the efficiency of FD and their robustness in real-time conditions. Rayi et al. on the other hand, used the Centroid Distance Fourier Descriptors as a hand shape descriptor. They implemented a system able to classify static hand gestures. For image acquisition they used a depth sensor, and hand segmentation was performed on the depth image with a threshold operation. A Canny edge detector was used to extract the hand contour. For hand gesture classification they used a similarity degree matching, where the smallest distance is considered as a match. For that, they evaluated the performance of different distance metrics as classifiers: Euclidean distance, Manhattan distance and Canberra distance. The experimental results showed that the Centroid Distance Fourier Descriptors with the Manhattan distance-based classifier achieved the best recognition rates, with an accuracy of 95% and with a small computational latency. Barczak et al. made experiments with Moment Invariants and Fourier Descriptors features for American Sign Language (ASL), where two important properties were covered: invariance to certain transformations and discrimination power. Three types of images were used with Moment Invariants: grey scale, binary and contours, and two different Fourier descriptors were compared, namely complex coordinates and centroid distance. Some problems arise from hand images that yield similar contours, because the correspondent ASL gesture only differs by the position of the thumb. In those cases, the Fourier Descriptors and the Moment Invariants computed would not be able to differentiate the classes.

Bourennane et al. [18] presented a shape descriptor comparison for hand posture recognition from video images, with the objective of finding a good compromise between accuracy of recognition and computational load for a real-time application. They run experiments on two families of contour-based Fourier descriptors and two sets of region based moments, all of them invariant to translation, rotation and scale-changes of hands. They performed systematic tests on the Triesch benchmark database [56] and on their own with more realistic conditions, as they claim. The overall result of the research showed that the common set Fourier descriptors when combined with the k-nearest neighbour classifier had the highest recognition rate, reaching 100% in the learning set and 88% in the test set.

Other types of features, like SIFT (Scale Invariant Feature Transform) [57] and SURF (Speeded Up Robust Features) [58] for example had also been used and tested with the same goal in mind: to find good features with invariant properties able to work in real-time conditions.

Wang et al. [59] proposed a system based on the discrete Ada-boost learning algorithm integrated with SIFT features, for accomplishing in-plane rotation invariant, scale invariant and multi-view hand posture detection. They used a sharing feature concept to increase the accuracy of multi-class hand posture recognition. The performances of hand detection using the Viola-Jones detector and the proposed approach were compared. The results of hand detection using Ada-boost with SIFT features were more satisfactory than with the Viola-Jones detector. With the implemented solution, they were able to successfully recognize three hand posture classes and deal with the problem of background noise. Huynh et al. [60] presented an evaluation of the SIFT (scale invariant feature transform), Colour SIFT (CSIFT), and SURF (Speeded Up Robust Features) descriptors on very low resolution images. The performance of the three descriptors was compared against each other on the precision (also called *positive predictive value*) and recall (also known as *sensitivity*) measures [61] using ground truth correct matching data. Their experimental results showed that the precision and the recall performances of SIFT and CSIFT were similar. Also, the experimental results showed that both SIFT and Colour SIFT were more robust under changes of viewing angle and viewing distance, but SURF was superior under changes of illumination and image blurring. In terms of computation time, the authors concluded that the SURF descriptors are a good alternative to SIFT and CSIFT.

Kolsch et al. [62] studied view-specific hand posture recognition system, where they used the Viola-Jones object recognition method [63] to detect the hand. The used method as the disadvantage of being computationally very expensive, prohibiting the evaluation of many hand postures. They introduced a frequency analysis-based method for instantaneous estimation of class separability, without the need for any training. They were also able to optimize the parameters of the detection method, achieving significant speed and accuracy improvements. Their study showed that, the Viola-Jones detector could achieve excellent detection rates for hand postures.

2.3 Machine Learning and Classification

Classification is generally achieved with a distance metric and nearest neighbour rules, but also with other classifiers or machine learning algorithms. Classification involves a learning procedure, for which the number of training images and the number of gestures are important facts [18]. Ian Millington et al. stated that making sure the learning has sensible attributes is part of the art of applying machine learning and getting it wrong is one of the main reasons of failure [64].

The study and computer modelling of learning processes in their multiple manifestations constitutes the topic of machine learning [65]. So, machine learning is the task of programming computers to optimize a performance criterion using example data or past experience [66]. Thereunto, machine learning uses statistic theory in building mathematical models, since its core task is to make inference from sample data.

In machine learning two entities, the teacher and the learner, play a crucial role. The teacher is the entity that has the required knowledge to perform a given task. The learner is the entity that has to learn the knowledge to perform the task. One can distinguish learning strategies by the amount of inference the learner performs on the information provided by the teacher. This way, the learning problem can be stated as follows: given an example set of limited size, find a concise data description [65]. Given this, learning techniques can be grouped in three big families: *supervised learning*, *reinforcement learning* and *unsupervised learning*.

In the current study, all the experiments and system implementations used supervised machine learning algorithms, and for that reason this state of the art focus on this type of algorithms only. In supervised learning, given a sample of input-output pairs, called the training sample, the task is to find a deterministic function that maps any input to an output that can predict future observations, minimizing the error as much as possible. According to the type of outputs, supervised learning can be distinguished in *classification* and *regression learning* [66]. In *classification* problems the task is to assign new inputs to one of a number of discrete classes or categories. If the output space is formed by the values of a continuous variable, then the learning task is known as the problem of *regression* or *function learning* [67].

Machine learning algorithms have been applied successfully to many fields of research like, face recognition [68], automatic recognition of a musical gesture by a computer [69], classification of robotic soccer formations [70], classifying human physical activity from on-body accelerometers [71], automatic road-sign detection [72, 73], static hand gesture classification [74], dynamic hand gesture classification, and speech recognition [75], among others.

As explained by Jain et al. [76], it is clear that no single approach for classification is *"optimal"*, depending for example on the nature and type of extracted features or the type of application. Consequently, it is a common practice sometimes to combine several modalities and classifiers for the purpose of pattern classification. In practice, the choice of a classifier is a difficult problem and it is often based on which classifier(s) happens to be available or known to the user.

Thus, among the various machine-learning algorithms used for gesture classification the following sections will describe the most representative for the current research work, in the field of static and dynamic gesture recognition, namely: *k-Nearest Neighbour* (k-NN), *Artificial Neural Networks* (ANN), *Support Vector Machines* (SVM) and *Hidden Markov Models* (HMM).

2.3.1 k-Nearest Neighbour (k-NN)

The *k-Nearest Neighbour* algorithm is one of the most simple and popular of all machine-learning algorithms. The algorithm simply compares a feature vector to be classified with all the features vectors in the training set with known class labels, and assigns the vector the most frequent class. To find the nearest neighbour a measure of distance is used. Euclidean distance is usually the preferred metric, but other metrics can be used as well, like the Mahalanobis distance, the Manhattan distance or the Canberra distance [77].

This type of classification although often with good results, has a number of drawbacks such as the need to define the number k (neighbours), which has direct impact on final performance of the algorithm. Also, the entire training set needs to be stored and used during classification, which affects performance with increasing size. On the other side, there are no restrictions on the distance metric to use.

By this rule, for *k = 3* for example, the nearest three neighbours are selected (those three with the least distance) and their mode, the maximally represented class, is assigned to the test sample. If *k = 1*, then the object is simply assigned to the class of its nearest neighbour.

The *k-Nearest Neighbour* algorithm for classification can be described as follows:

Algorithm 4. k-NN classification algorithm

```
inputs:
  D = {($\mathbf{x}_1$, $c_1$), …, ($\mathbf{x}_N$, $c_N$)}: vectors with known classes
  x = ($x_1$, …, $x_N$): new instance (vector) to be classified

begin
  for each labelled instance (xi, ci) do
    calculate d($x_i$, x), the distance from the new instance to (xi, ci)
  endfor

  for i=1 to N do
    order d(xi, x) from lowest to highest
  endfor

  select the k nearest instances to x: $(\boldsymbol{D}_x^k)$
  assign to x the most frequent class in $\boldsymbol{D}_x^k$
end
```

2.3.2 Artificial Neural Networks (ANN)

An Artificial Neural Networks is a system that attempts to model the way the human brain works. It comprises a network of artificial neurons, which are mathematical models of biological neurons. Like the biological neuron, an artificial neuron (called a perceptron), receives numerical values and outputs a numerical value. There are three types of neurons in an ANN – *input, hidden* and *output* neurons, as can be seen in Figure 2, which are connected in a special pattern known as the neural network's architecture or topology. In this type of architecture (multi-layer perceptron), each node takes input from all the nodes in the preceding layer and sends a single output to all the nodes in the next layer. For that reason it is called a *feedforward network.*

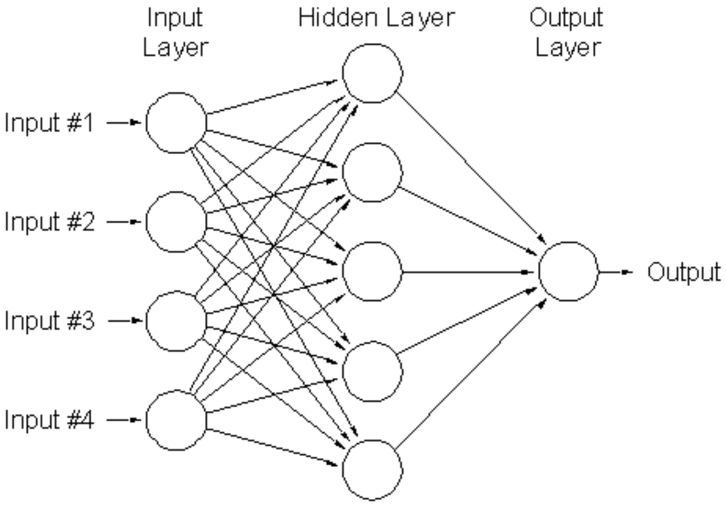


Figure 2. Artificial Neural Network [78]

During classification, the input or leftmost layer into the perceptron is provided with a numerical value (activation value provided by the system) that is multiplied by a weight (connection strengths). The perceptron only fires an output, the rightmost layer, when the total strength of the input signals exceeds a certain threshold.

The weighted input to a perceptron is acted upon by a function, the transfer function, which will determine the activation output (Figure 3). Common transfer functions used in artificial Neural Networks include the unit step (equation 10), sigmoid (equation 11) and Gaussian (equation 12). The activation flows through the network through the hidden layers, until it reaches the output nodes.

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{10}$$

$$f(x) = 1/((1 + e^{\beta x})\,) \tag{11}$$

$$f(x) = 1/\left(\sqrt{2\pi\sigma}e^{-(x-\mu)^2/2\sigma^2}\right) \tag{12}$$

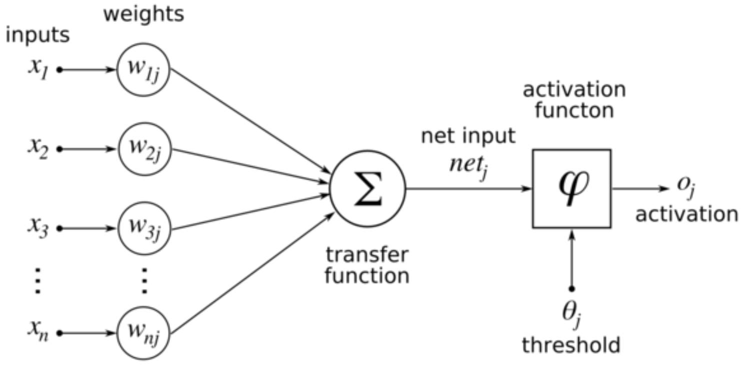


Figure 3. Neural network with inputs, activation function and output represented. [79]

For learning, the neural network has to be put in a specific learning mode. Here another algorithm applies, the learning rule, which although using the original perceptron algorithm, it is more complex and the most common algorithm used is the *backpropagation* [64]. This algorithm works in the opposite direction of the *feedforward algorithm*, working backwards from the output. Additional information can be found in the books by Haykin [80] and by Gorunescu [81].

2.3.3 Support Vector Machines (SVM)

Support vector machines are a group of supervised learning methods that can be applied to classification or regression. SVM's select a small number of boundary feature vectors, *support vectors,* from each class and builds a linear discriminant function that separates them as widely as possible (Figure 4) - *maximum-margin hyperplane* [77]. Maximum-margin hyperplanes have the advantage of being relatively stable, i.e., they only move if training instances that are support vectors are added or deleted. SVM's are non-probabilistic classifiers that predict for each given input the corresponding class.

In the two-class example, a set of training samples (feature vectors), each marked as belonging to one of the classes is used to learn a model that will be used in the classification phase.

In addition to performing linear classification, SVM's can efficiently perform non-linear classification using what is called the kernel trick [82], implicitly mapping their inputs into higher-dimensional feature spaces. Mathematically, any function $K(x, y)$ is a kernel if it can be written as $K(x, y) = \Theta(x) \cdot \Theta(y)$, where Θ is the function that maps an instance into the higher-dimensional feature space. So, the kernel function represents a dot product in the feature space created by Θ. Additional information can be found in the books by Theodoridis [83] and by Alpaydin [66].

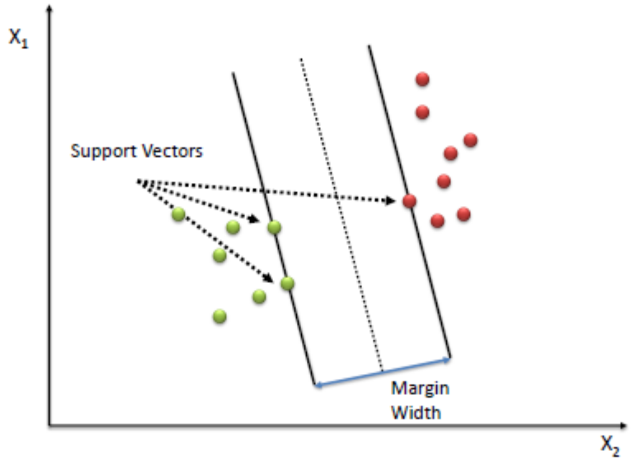


Figure 4. SVM: support vectors representation with maximum-margin hyperplane [84]

2.3.4 Hidden Markov Models (HMM)

The typical model for a stochastic (i.e. random) sequence with a finite number of states is called a Markov Model [85]. When the true states of the model $S = \{s_1, s_2, s_3, \dots, s_N\}$ are hidden in the sense that they cannot be directly observed, the Markov model is called a Hidden Markov Model (HMM). At each state an output symbol $O = \{o_1, o_2, o_3, \dots, o_N\}$ is emitted with some probability, and the state transitions to another with some probability, as shown in Figure 5. With discrete number of states and output symbols, this model is sometimes called a "*discrete HMM*" and the set of output symbols the *alphabet*.

In summary, an HMM has the following elements:

- **N**: the number of states in the model $S = \{S_1, S_2, \dots, S_N\}$;
- **M**: the number of distinct symbols in the alphabet $V = \{v_1, v_2, \dots, v_M\}$;
- State transition probabilities:
  $\boldsymbol{A} = [a_{ij}]\ where\ a_{ij} \equiv P(q_{t+1} = S_j | q_t = S_i)\ and\ q_t\ is\ the\ state\ at\ time\ t$;

- Observation probabilities:
  $\boldsymbol{B} = \{b_j(m)\}\ where\ b_j(m) \equiv$
  $P(O_t = v_m | q_t = S_j)\ and\ O\ is\ the\ observation\ sequence$;
- Initial state probabilities: $\boldsymbol{\Pi} = [\pi_i]\ where\ \pi_i \equiv P(q_1 = S_i)$;

and is defined as $\lambda = (A, B, \Pi)$, where N and M are implicitly defined in the other parameters. The transition probabilities and the observation probabilities are learned during the training phase, with known gesture data, which makes this is a supervised learning problem [8].

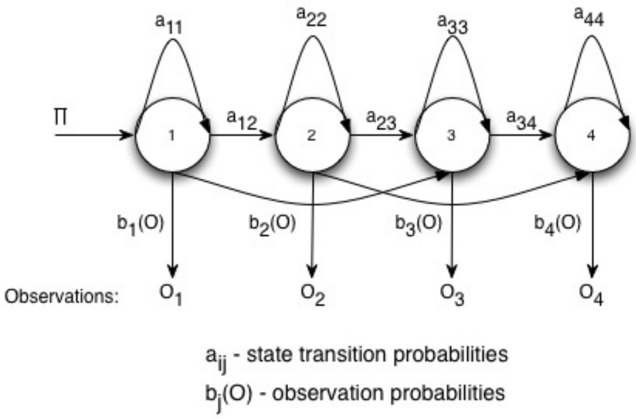


Figure 5. HMM example with initial probabilities (**Π**), transition probabilities (a_{ij}), observation probabilities ($b_j(m)$) and observation sequence (O) represented.

2.3.5 Related Work

As explained in section 2.3 no single approach for classification is *"optimal"*, depending for example, on the nature and type of extracted features or the type of application. Consequently, as seen before, it is sometimes a common practice to combine several modalities and classifiers for the purpose of pattern classification. In practice, the choice of a classifier is a difficult one and it is often based on which classifier(s) happen to be available or known to the user.

2.3.5.1 k-Nearest Neighbor

Being the k-NN algorithm one of the simplest and popular of all machine-learning algorithms, normally used in conjunction with a distance metric many authors have used it in their implementations. Yang Jufeng et al. [86] implemented a command system based on static hand gesture recognition for controlling a PowerPoint presentation that used a k-NN classifier. The hand was detected based on the HSV (Hue-Saturation-Value) model and made use of Fourier descriptors as hand features. In the experiments the authors used a 4-gesture database with 400 records, divided into a train and a test dataset. On the train dataset they achieved an accuracy of 95% for the first gesture, 90% for the second, 88,3% for the third and 86,7% for the last one. For the test dataset, they obtained a high error rate on the first and third gestures, resulting from light conditions on the image and on the quality of hand posture image obtained. Also, their system had some limitations on cluttered backgrounds and with varying hand angles.

Bourennane et al. [18] performed experiments on two sets of contour-based Fourier descriptors and two sets of region based moments with a k-NN classifier. The overall result of the research showed that the common set Fourier descriptors when combined with the k-nearest neighbour classifier had the highest recognition rate, reaching 100% in the learning set and 88% in the test set. Hanning Zhou et al. [42] applied a k-NN classifier with a Euclidean distance metric, for hand posture recognition on a dataset composed of mean vectors obtained by clustering a set of augmented normalized HoG features. The experiments showed that the purposed method was able to capture the distinct hand shape without requiring clean segmentation. Also, they were able to achieve higher recognition rates when compared to other techniques, and with a better efficiency in terms of computationally speed. Mohamed Bécha Kaâniche et al. [48] introduced a new HoG tracker for gesture recognition based on local motion learning which used a k-NN classifier in the learning/recognition phase. They have validated the local motion descriptors with the KTH database [49], with which were able to achieve an accuracy of 91,33% when using the Shi-Tomasi corner detector [29] (section 2.2.6), and an accuracy of 94,67% when using the FAST (Features from Accelerated Segment Test) corner detector [50].

Herve Lahamy et al. [87] performed a comparative analysis of hand features and classifiers for automatic recognition of eight different gestures, using datasets collected with a range camera. The features included Hu-moments, orientation histograms and hand shape associated with its distance transformation image. As classifiers, the k-nearest neighbour algorithm (k-NN) and the chamfer distance had been chosen. For an extensive comparison, four different databases have been collected with variation in translation, orientation and scale. The evaluation was performed by measuring the separability of classes, and by analysing the overall recognition rates as well as the processing times. The best result were obtained from the combination of the chamfer distance classifier and hand shape and distance transformation image, but the time analysis revealed that the corresponding processing time was not adequate for a real-time recognition.

2.3.5.2 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANNs) have also been widely used in the area of vision-based gesture recognition with very good results. Maung [88], Tasnuva Ahmed [89] and Mekala et al. [90] applied Artificial Neural Networks (ANNs) to the problem of static hand gesture recognition in a real-time system. Maung applied it to a system able to recognize a subset of MAL (Myanmar Alphabet Language) static hand gesture in real time. His method extracted HoG features that were used to train an ANN classifier. The paper includes experiments with 33 hand postures, and the author claims that the method is simple and computationally efficient. The results showed that the system was able to achieve an accuracy of 98% on a single hand database. One drawback of the method is, that it is efficient as long as the data sets are kept small. Tasnuva Ahmed trained the ANN with a database of records composed of 33 different features extracted from grey scale and binary hand images, that the author claims are rotation, scaling, translation and orientation independent. The ANN architecture used had an input layer with 33 inputs, an hidden layer with 85 nodes or neurons, and an output layer composed of 4 nodes, and used the back-propagation learning algorithm. The proposed approach was able to achieve an accuracy of 88,7% with a database composed of four static gesture types. The system presented however some difficulties in varying light conditions and hand tracking. Mekala et al. on the other side, used an ANN in a vision-based system for static hand Sign Language recognition, based on a HW/SW co-simulation platform. This approach intends to increase the speed of execution, while maintaining the flexibility. As feature vectors they used the extracted hand shape and the orientation magnitude. The training phase is done on the software platform and the testing phase is done on the hardware platform. This is based on a new technique presented in the paper called *co-simulation neural network*. In their method, a part of the neural network is designed on the hardware with dedicated ports. The network is built of 16 input neurons in the input layer, 50 neurons in the first hidden layer, 50 neurons in second hidden layer, and 35 neurons in the output layer. The number of neurons selected resulted from the analysis of the selected features for image representation. The authors claim that the co-simulation platform was able to reduce the recognition times, with a 100% success rate for all the sign language alphabets (A to Z). Haitham Hasan et al. [91] presented a novel technique for static hand gesture recognition for human-computer interaction based on the hand shape analysis and an ANN for hand gesture classification. Their main goal was to explore the utility of a NN approach for the recognition of hand gestures. The system was divided into three parts: (1) pre-processing, (2) feature extraction and (3) classification. The main efforts were made in the feature extraction and classification. As hand features they used complex moments (CMs) introduced by Abu-Mostafa [92] as a simple and straightforward way to derive moment invariants. The CMs are very simple to compute and quite powerful in providing an analytic characteristic for moments invariant. They are a set of values extracted from the image that have the property of invariance to image rotation. Moment invariant can be used as a feature for object classification and recognition. They tested a Neural Network (NN) with the extracted hand contour and a Neural Network with the complex moments. The system was able to achieve a recognition rate of 70,83% using the contour but suffered from invariance to translation, while the second method achieved an accuracy of 86,38%. Nevertheless, the system presented some problems to recognize the same gesture under different light conditions, which is a serious limitation

2.3.5.3 Support Vector Machines (SVM)

Being Support Vector Machines (SVMs) a group of supervised learning methods that can be applied to classification or regression, and in addition to performing linear classification, can efficiently perform non-linear classification using what is called the kernel trick (section 2.3.3), they have been widely applied to the problem of hand gesture recognition. Ching-Tang Hsieh et al. [93] presented a fast hand detection and an effective feature extraction process for a real-time hand gesture recognition system based on a SVM classifier. The system was comprised of four steps: (1) the use of the Camshift Algorithm (Continuously Adaptive Mean Shift) for skin colour tracking, primarily intended to perform efficient head and face tracking in a perceptual user interface [94], (2) the use of the Boundary Extraction Algorithm (BEA), proposed by Liu [95], to extract the hand contour, (3) extraction of Fourier Descriptors (FDs) for shape description and (4) shape classification using a SVM. As shown in section 2.2.5, Fourier descriptors have some advantages, namely invariance to starting boundary point, deformation and rotation. For the problem of hand gesture recognition with different view angles, the authors acquired multiple images. Each gesture contained 1000 images with a total of 5000 images for all the gestures. The experimental results showed they were able to achieve an average accuracy of 93,4%, demonstrating their system was feasible. Yen-Ting Chen et al. [96] presented a system that allows effective recognition of multiple-angle hand gestures in finger guessing games. They used three webcams set at front, left and right directions of the hand for image acquisition. By using images from three different cameras, three SVMs classifiers were trained. After the training process, the constructed classifiers were fused, according to three different plans. In the first, with simple voting, each classifier contributes equally to the final result, while in the second and third plans, the recognition rates were regarded as fusion factors, i.e. the recognition rate obtained with one of the cameras for one gesture was given as a fusion factor. As the experimental results had shown, the fusion strategy made the recognition system more robust and reliable, being able to achieve an accuracy of 93,33% in terms of multi-view hand gesture recognition. Jinbing Gao et al. [53] trained a SVM with features obtained by fusing HoG features and LBP features to track the palm in unfettered colour images. They called the proposed method Adaptive HoG-LBP detector. They defend that the object description ability of the HoG and the LBP are different, so first, they determine the strength of each descriptor on the hand image or part of the image based on what they call a Confidence Map Computing. The extracted features are then fused based on the calculated confidence values and are normalized. They built their own dataset with 5378 palm images from 28 different users, and the experiments showed that they were able to achieve an accuracy of 95,2% in real-time situations. The classification is done without any pre-processing such as noise filtering, brightness balance or sharpening. The system showed however some problems with hand motion detection, which was defined as future work. Liu Yun t al. [97] presented an automatic hand gesture recognition system that consisted of two modules: a hand detection module and a gesture recognition module. For hand detection, they used the Viola-Jones method [63]. For hand recognition and SVM train, they used Hu invariant moments [55] of the hand image as feature vectors. Hu invariant moments have the advantage of being invariant to translation, scale and rotation. To overcome the linearity of the basic SVM classifier the authors used the on-vs-all approach, which separates a single class from all the remaining classes. The experiments showed that the system were able to achieve a total recognition accuracy of 96,2% for the three hand postures defined in the dataset. However, the system showed some problems with test images acquired under strong light conditions, which led to some incorrect results. The authors had concluded, that in those cases, the failure mainly stems from erroneous segmentation of background areas as belonging to hand regions. Jung-Ho Ahn et al. [98] presented a real-time colour based tracking method, which they called the hand motion vectors. The system was able to track both hands and was based on the mean-shift (MS) tracking algorithm. They assumed that a gesture was performed in front of a camera, and the starting and end postures are the same. For the occlusion recovery problem, they presented a tracking method that used a predicted window, i.e. the candidate hand window position was predicted by using past motion information. As hand features the authors used a simple and efficient feature extraction method. They stated that all hand trajectories are not equally important to identify a gesture, so, by singular state analysis they filtered out the trivial trajectories. For that, they defined three states of hand trajectories with their respective mathematical descriptions: *hold state*, *dynamic state* and *occlusion state*. With those definitions a motion vector would be classified as singular or normal. After feature extraction, motion vectors were normalized and a quantization process transformed a sequence of motion vectors into a set of symbols (codewords). A vector quantizer consists of a codebook and a quantization function, that in the proposed system simple mapped the motion vectors to the nearest codeword. For gesture classification they used a SVM, but to perform SVM classification first they had to make sure that all the features were in the same space, so they had to make vectors of features with constant length. Unfortunately, as the experimental results showed, they had very high error rates, mainly in two gestures that were very similar, and the SVM recognition rate was not so high. The problem stemmed from very noise data, where several states appear in several features. Nasser H. Dardas et al. [99] presented a system for real-time interaction with an application or videogame via hand gestures. The system was able to detect and track a bare hand in cluttered backgrounds using skin detection. To remove other skin-like areas, they first detect the face with the Viola-Jones method, which was replaced by a black circle. After face subtraction, other skin areas are detected using the HSV colour model. The contours of skin areas are extracted and compared with all hand gestures contours loaded from a dataset, to get rid of non-hand skin-like objects still present in the image. As hand features, they used key points extracted with the SIFT (scale invariant feature transform) algorithm. Since the obtained features were too high dimensionality to be used efficiently, the authors solved the problem with the bag-of-features approach [100, 101] to reduce the dimensionality of the feature space. This way, each training image can be described as a “bag-of-words” vector by mapping key points to a vector of visual words. With the obtained feature representation a multi-class SVM was trained. The testing stage proved that effectiveness of the proposed scheme in terms of accuracy and speed. Experiments showed that the system could achieve satisfactory real-time performance regardless of the frame resolution and with an accuracy of 96,23% under variable scale, orientation and illumination conditions and also with cluttered backgrounds. The authors identified three important factors that could affect the accuracy of the system, namely the quality of the webcam used in the training and testing stages, the number of training images and choosing the number of clusters used as codebook size. Chen-Chiung Hsieh et al. [102] proposed a real-time hand gesture recognition system that consisted of three major parts: digital zoom, adaptive skin detection and hand gesture recognition. The first module detected the user's face, and applied trivial trimming and bilinear interpolation for zooming in, so that the face and upper body occupied the central part of the image. The second module was based on an adaptive skin colour model for hand region extraction. By adaptive skin colour model, the effects from lighting, environment, and camera could be greatly reduced, and the robustness of static hand gesture recognition could be improved. A ROI was defined under user’s face, and a three level grey-level Haar-like feature was used to extract the hand region. For dynamic gestures, they observed the motion history image (MHI) for each dynamic directional hand gesture and designed four groups of Haar-like directional patterns able to recognize up, down, left and right movements. The benefit of the motion history image was that it could preserve object trajectories in a frame. Those patterns count the number of black-white patterns as statistical features for classification. They investigated the use of a Neural Network for dynamic gesture recognition, but found that the accuracy rate stopped at about 85% because some variations of different dynamic hand gestures were similar. To overcome this problem, they investigated a new approach based on a SVM for dynamic hand gesture classification. The overall system was able to obtain a recognition rate of 93,13% for dynamic gestures and 95,07% for static hand gestures in average.

2.3.5.4 Hidden Markov Models (HMM)

Since dynamic gestures are time-varying processes, which show statistical variations, HMMs are a plausible choice for modelling them. In this way, a human gesture can be understood as a HMM where the true states of the model are hidden in the sense that they cannot be directly observed. Many authors have used HMMs for gesture recognition.

Chen et al. [103] introduced a hand gesture recognition system to recognize continuous gestures with a stationary background. The system is composed of four modules: a real-time hand tracking and extraction module, a feature extraction module, and an HMM training module and gesture recognition module. First, they applied a real-time hand tracking and extraction algorithm to trace the moving hand

and extract the hand region, and then they used Fourier Descriptors to characterize spatial features and motion analysis to characterize temporal features. They combined the spatial and temporal features of the input image sequence as the feature vector. During the recognition phase, the extracted feature vector was separately scored against different HMMs. The model with the highest score indicated the corresponding gesture. They have tested the system with a dataset built with 20 different gestures obtained with different users, and they were able to achieve an accuracy of 85%. They had some problems with wrong gesture identification that they claim have originated in the limited number of records used to estimate the parameters of the HMMs. Yoon et al. [104] proposed a system consisting of three different modules: (1) hand localization, (2) hand tracking and (3) gesture spotting. The hand location module detects hand candidate regions, on the basis of skin colour and motion. The hand-tracking algorithm calculates the moving hand regions centroids, connects them, and outputs a hand path. The gesture-spotting algorithm divides the trajectory into real and meaningless segments. To construct a feature database they used a combined and weighted location, angle and velocity feature codes, and employed a k-means clustering algorithm for the HMM codebook. In their experiments, 2400 trained gestures and 2400 untrained gestures were used for training and testing, respectively. They were able to obtain an accuracy of 93% in a batch test and an accuracy of 85% on the on-line test that used direct camera input and real time HMM recognition.

Nguyen Dang Binh et al. [105] introduced a real-time gesture recognition system, for single hand gestures, that could be used in unconstrained environments. The system was composed of three modules: hand tracking, gesture training and gesture recognition using pseudo two dimension Hidden Markov Models (P2-DHMMs), introduced and applied by Agazzi and Kuo to the problem of optical character recognition [106]. The P2-DMHH uses observation vectors that are composed of two-dimensional Discrete Cosine Transform (2-D DCT) coefficients. Their main contribution was the combination in the P2-DHMM framework of the hand region information and the motion information. They tested the system using the American

Sign Language gestures. The training data was composed of 30 images for each one of the 36 gestures. They were able to obtain an overall accuracy of 98%.

Mahmoud Elmezain et al. [107] proposed an automatic system based on Hidden Markov Models and that is able to recognize both isolated and continuous gestures for Arabic number (0-9) recognition in real time. To handle isolated gestures, three different HMM topologies (Ergodic, Left-Right and Left-Right Banded) with different number of states were tested. Hand skin segmentation is done with a Gaussian Mixture Model applied to stereo colour images with complex backgrounds. From the obtained hand they extracted three basic features: *location*, *orientation* and *velocity*. The orientation is quantized by dividing the value by 20º to generate code words that are between 1 and 18. The discrete vector thus obtained is used as input to the HMM. For continuous gesture recognition, the system is designed to segment and recognize an isolated gesture by what the authors called the zero-code word detection. Although some gestures contain the zero-code word in some segment parts, they used a static velocity threshold to overcome the problem of incorrect classifications or gesture separation. Also, in order to take into account the transition between gestures, the system ignores some links between the two gestures by neglecting some frames adaptively after detecting the gesture end point. In their experiments they used 30 video sequences for isolated gestures and 70 video sequences for continuous gestures. They were able to achieve an average accuracy of 98,94% for the isolated gestures and average accuracy of 95,7% for the continuous gestures.

Omer Rashid et al. [108] proposed an interaction system through gesture and posture recognition for alphabets and numbers. The gesture system was able to recognize the hand motion trajectory using HMM whereas the posture system classifies the static hand at the same instance of time. In the proposed system, 3D information is exploited for segmentation and detection of face and hands using normal Gaussian distributions and depth information. For the gestures, they compute the orientation of two consecutive hand centroids, for all the points in the hand path, which is then quantized in the range 1 to 18, in order to generate the code words. The obtained quantized values gives a discrete vector that is used to train the HMM. For the problem of posture recognition, feature vectors are computed from statistical and geometrical properties of the hand. As statistical features they used Hu-Moments (area, mean, variance, covariance and skewness), and as geometrical features the authors used the circularity and rectangularity in order to exploit the hand shape. These features are then combined together to form a feature set which is used to train a SVM for classification and recognition. The experimental results of the proposed framework could successfully integrate both gesture and posture recognition. For the gesture system a recognition rate of 98% was achieved (alphabets and numbers) and for the posture system a recognition rate of 98,65% and 98,6% for ASL alphabets and numbers was achieved respectively.

Sara Bilal et al. [109] provide a good survey on approaches which are based on Hidden Markov Models (HMM) for hand posture and gesture recognition for HCI applications. Their main goal was to provide a survey of HCI applications using hand gestures which have been developed based on vision systems using HMM's. In their paper they gave a brief introduction to the computational tools used to manipulate HMM's, they presented HCI applications that have been developed using HMM for hand posture and/or gesture recognition and they made a comparison of HMM with other existing methods for hand posture and/or gesture recognition techniques. The authors had concluded in the paper that most of the developed systems have been designed for a certain HCI application and might not achieve the same declared accuracy for another application, being this dependent on many factors such as: using data gloves or bare hands, the number of database samples, isolated words or sentence level recognition in case of SL (Sign Language) and using single hand or two hands.

2.3.5.5 Others

Many authors have presented studies were they tested different classifiers, for the problem of classification performance, or even used different classifiers on different parts of their systems with different goals in mind. Anand H. Kulkarni et al. [110] presented a robust hand gesture recognition system for static gesture classification based on 11 Zernike moments (ZMs) [111] and tested the extracted features with three different classifiers: the k-NN (k-Nearest Neighbour), the ANN (Artificial

Neural Network) and the SVM (Support Vector Machine). Zernike moments have the advantage of being rotation and scaling invariant. A comparative study was carried out to test which classifier performed better in recognizing the pre-defined set of gestures. The SVM was the classifier that obtained the best results with an accuracy of 91% compared to 77,5% for the k-NN and 82,5% for the ANN. Avilés et al. [112] made an empirical comparison of three classification techniques, namely, *Neural Networks*, *Decision Trees* and *Hidden Markov Models*, focused not only on the problem of classification performance in gesture recognition, but also trying to find the best in respect to knowledge description, feature selection, error distribution and computational time for training. The results showed that none of the techniques is a definitive alternative for all the questions addressed in the study. They used a gesture database with more than 7000 samples performed by 15 users. While Neural Networks and Hidden Markov Models obtained higher recognition rates in comparison to Decision Trees, they claim that the knowledge description of decision trees allows them to analyse interesting information suck as the similarity of gestures. Also, due to the required computational time for training, decision trees could be adequate for fast prototyping gesture recognition interfaces for HCI (*Human Computer Interaction*). Average recognition rates for their experiments, with a database built from 15 people making 9 different dynamic gestures were, 95,07% for the ANN, 94,84% for the HMM and 87,3% for the DT's. Omer Rashid et al. [108] presented a system for ASL (American Sign Language) gesture (dynamic) and posture (static) recognition of alphabets and numbers able to provide interaction through. In their system, they have exploited 3D information for segmentation and detection of face and hands using normal Gaussian distribution and depth information. For gesture, orientation of two consecutive hand centroid points is computed which is then quantized to generate code words. HMMs were trained with the Baum Welch algorithm and classification was done with the Viterbi path algorithm. Feature vectors were computed from statistical and geometrical properties of the hand, namely, Hu-Moments, circularity, rectangularity and fingertips. After normalization, the extracted features were then used to train a SVM, used later for classification and recognition. Experimental results showed that the proposed framework was able to successfully integrate both the gesture and posture recognition systems, where the gesture recognition system achieved a recognition rate of 98% (for alphabets and numbers) and the posture recognition system achieved a recognition rate of 98,65% and 98,6% for alphabets and numbers respectively. Oshita et al. [113] on the other hand proposed a general gesture recognition system method, based on two different machine-learning algorithm: the *Self-Organizing Map* (SOM), developed in 1982 by Tuevo Kohonen [114], and the *Support Vector Machine* (SVM) [115]. Their system could handle any kind of input data from any input device. In the experiments they used two Nintendo Wii Remote controllers, with acceleration sensors. The SOM is used to divide the sample data into phases and construct a state machine. Then, they apply the SVM to learn the transition conditions between nodes. An independent SVM is applied at each node. They tested the method with two kinds of gestures: a simple one, performed with just one controller, and a more complex one done with the two controllers. Their method showed some problems in the presence of noise, since the classification using SOM does not work well in those situations. Because of this, automatic gesture recognition was not able to achieve a good recognition rate.

Other methods were also tried out with good results, like the one by Bailador et al. [116], were an approach to the problem of real-time gesture recognition using inexpensive accelerometers was presented. It is based on the idea of creating specialized signal predictors for each gesture class. The errors between the measured acceleration of a given gesture and the predictors are used for classification. The predictors were implemented using Continuous Time Recurrent Neural Networks (CTRNN), which are networks of continuous model neurons without constraints placed on their connectivity [117], and that exhibit rich dynamics [118]. The dynamic and non-linear nature of the CTRNN makes them suited for temporal information processing. They used a set of eight different gestures to test the performance and accuracy of the recognition method. Two different datasets were recorded, with gestures made by only one person in different condition and with twenty instances for each gesture, resulting in a total of 160 gesture instances per dataset. With this, they were able to obtain a recognition rate of 98% for the training set and 94% for the testing set.

2.4 Summary

In this chapter the methodologies normally used for vision-based hand gesture recognition addressed. Some of the methods studied and implemented for the problem of hand feature extraction, posture classification and dynamic gesture classification were described.

In terms of hand features for gesture classification, it was seen that many possibilities exist with different results, and careful feature selection is vital for the future success of the recognition system. Also, efficient features must present some essential properties like: translation, rotation and scale invariance, occlusion invariance, noise resistance and be reliable.

A review of the state of the art in the area was conducted, which showed clearly the variety of experiments done so far in the field, with or without combinations of features, and where we could note that there is no single solution for the problem at the moment.

In terms of gesture classification, some of the most used machine learning algorithms were described. We saw that in practice, the choice of a classifier is a difficult problem, and that no single approach is '*optimal*' depending on the nature and type of extracted features or the type of application. A review of the state of the art in this area was also undertaken, where we could note that it is sometimes common practice to combine several classifiers with the goal of obtaining a better pattern classification.

Despite all the work done in the area so far, we can conclude that there is still a long way to go in order to achieve a natural gesture-based interface for human/computer communication and interaction. Each possible contribution is a step forward in the attempt to reach such a solution.

Chapter 3

3 Gesture Learning Module Architecture (GeLMA)

3.1 Introduction

As analysed in the previous chapter, vision-based hand gesture recognition is an area of active current research in computer vision and machine learning [88]. Being a natural way of human interaction, it is an area where many researchers are working on, with the goal of making human computer interaction (HCI) easier and natural, without the need for any extra devices [74, 119].

As Hasanuzzaman et al. [120] argue, it is necessary to develop efficient and real time gesture recognition systems, in order to perform more human-like interfaces between humans and robots.

Although it is difficult to implement a vision-based interface for generic usage, it is nevertheless possible to design this type of interface for a controlled environment [10, 121]. Furthermore, computer vision based techniques have the advantage of being non-invasive and based on the way human beings perceive information from their surroundings [8].

Vision-based hand gesture recognition systems have a wide range of possible applications [13, 18], of which some are here highlighted:

- ***Virtual reality***: enable realistic manipulation of virtual objects using ones hands [122, 123], for 3D display interactions or 2D displays that simulate 3D interactions.
- ***Robotics and Tele-presence:*** gestures used to interact with robots and to control robots [6] are similar to fully-immersed virtual reality interactions, however the worlds are often real, presenting the operator with video feed from cameras located on the robot. Here, for example, gestures can control a robot's hand and arm movements to reach for and manipulate actual objects, as well as its movement through the world.
- ***Desktop and Tablet PC Applications***: In desktop computing applications, gestures can provide an alternative interaction to mouse and keyboard [124-

127]. Many gestures for desktop computing tasks involve manipulating graphics, or annotating and editing documents using pen-based gestures.

- ***Games***: track a player's hand or body position to control movement and orientation of interactive game objects such as cars, or use gestures to control the movement of avatars in a virtual world. Play Station 2 for example has introduced the Eye Toy [128], a camera that tracks hand movements for interactive games, and Microsoft introduced the Kinect [4] that is able to track users full body to control games.
- ***Sign Language***: this is an important case of *communicative gestures*. Since sign languages are highly structural, they are very suitable as test-beds for vision-based algorithms [11, 19, 129, 130].

However, to be able to implement such systems, there are a number of requirements that the system must satisfy, in order to be implemented in a successful way [10], which are:

- ***Robustness***: the system should be user independent and robust enough to factors like visual noise, incomplete information due for example to occlusions, variations of illumination, etc.
- ***Computational efficiency***: vision based interaction requires real-time systems, so the algorithms and learning techniques should be the most effective possible and computational cost effective.
- ***Error tolerance***: mistakes on vision-based systems should be tolerated and accepted. If some mistake is made, the user should be able to repeat the command, instead of letting the system make wrong decisions.
- ***Scalability***: the system must be easily adapted and configured so that it can serve a number of different applications. The core of vision based applications for human computer interaction should be the same, regardless of the application.

In order to be able to implement a vision-based solution that can be generic enough, with the help of machine learning algorithms, allowing its application in a wide range of interfaces for online gesture recognition, we need to have systems that allow training gestures and learning models capable of being used in real-time interaction systems. These systems should be easily configurable in terms of the number and type of gestures that they can train, to ensure the necessary flexibility and scalability.

In order to build a solution that could meet all the previous requirements, three modules were implemented. These modules allow you to train, in a supervised way, all the necessary gestures that will be part of future vision-based hand gesture recognition systems, for human / computer interaction.

The implemented modules are: the *Pre-Processing and Hand Segmentation* (**PHS**) module, the *Static Gesture Interface* (**SGI**) module and the *Dynamic Gesture Interface* (**DGI**) module as shown in the diagram of Figure 6. From the diagram, one can see that user hand must be detected, tracked and segmented on each frame. The segmented hand is passed as an argument to the SGI module to extract hand features that are saved into a features dataset. This dataset is later used for model training, and the resulting model is also for future use. The detected hand is passed as a parameter to the DGI module, where the hand centroid is calculated and used for hand path construction. Each hand path is labelled according to a defined alphabet resulting in a feature vector that is saved into the gesture dataset. This dataset is later used for model training and the corresponding obtained model is saved. The user has also the possibility to define the final commands that the system will be able to interpret in the command language definition module.

For the *Command Language Definition* (**CLD**) module, only a simple version was implemented able to be integrated with the final framework, being the final version implementation planned for further work.

The system uses only one camera, and is based on a set of assumptions, hereby defined:

1. The user must be within a defined perimeter area, in front of the camera.
2. The user must be within a defined distance range, due to camera limitations. The system defined values are 0.7m for the near plane and 3m for the far plane.
3. Hand pose is defined with a bare hand and not occluded by other objects.

4. The system must be used indoor, since the selected camera does not work well under sun light conditions.

The following sections describe in detail each one of the proposed modules. First, it is described the **PHS** (Pre-Processing and Hand Segmentation) module, where the problem of hand detection and tracking is addressed, as well as the problem of hand segmentation. Secondly, it is described the **SGI** (Static Gesture Interface) module, responsible for training static gestures and learn the model for a set of predefined hand postures, and finally it is described the **DGI** (Dynamic Gesture Interface) module, which is responsible for the dynamic gesture training, creating one model for each one of the predefined gestures to be used.

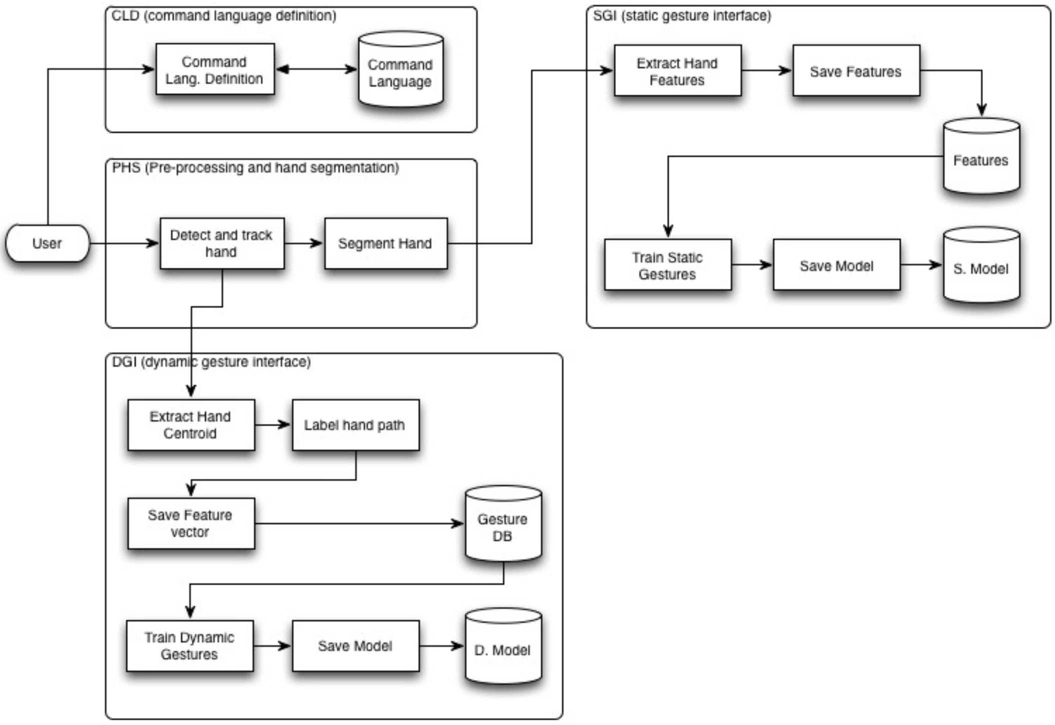


Figure 6. The static and dynamic gesture training and learning architecture.

3.2 Pre-Processing and Hand Segmentation (PHS) module

The Pre-Processing and Hand Segmentation module is responsible for hand detection, tracking and segmentation, trying to minimize the effects of noise present in depth image as explained in section 3.2.2.

Pre-processing and feature extraction plays an important role on the rest of system, namely, gesture recognition or classification. The approach used in the study is based on appearance-based methods [11, 18, 119]. Although viewpoint dependent, this type of methods is computationally efficient. They model gestures by relating the appearance of a given gesture to the appearance of a set of template gestures. The approach mainly consists of extracting a set of features that represent the content of the images [18]. Thus, good hand segmentation and proper choice of visual features are vital for the future performance of the recognition system.

Taking this into consideration, and to get a better understanding of the steps taken in this phase, this section is divided into two parts: (1) *hand detection and tracking*, and (2) *hand segmentation*.

3.2.1 Hand Detection and Tracking

For hand feature extraction, first the hand must be detected and tracked as shown in the diagram of Figure 7, under the PHS (Pre-Processing and Hand Segmentation) module, and the hand segmented for later use.

To solve this problem a Kinect camera with the OpenNI framework interface [131] is used, that is able to detect and track the hand position in space in real-time, as intended shown in Figure 8. OpenNI is a non-profitable consortium formed to promote and standardize the compatibility and interoperability of Natural Interaction (NI), devices, applications and middleware [131].

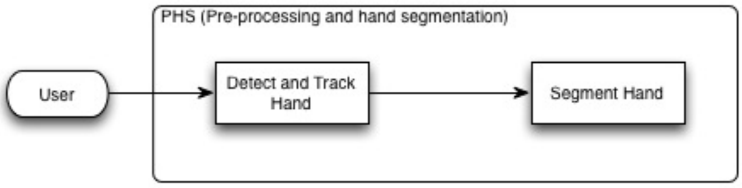


Figure 7. Pre-processing and feature extraction diagram.

The Kinect has a depth sensor consisting of an infrared laser projector combined with a monochrome CMOS sensor - an active pixel sensor - (Figure 9 and Figure 10), which is able to capture video data in 3D under any ambient light conditions [132]. The camera returns a depth image, updated 60 times per second according to Primesense [133], as shown in Figure 11 where different depth values are represented with different grey values. This is considered a great advantage over other type of cameras.

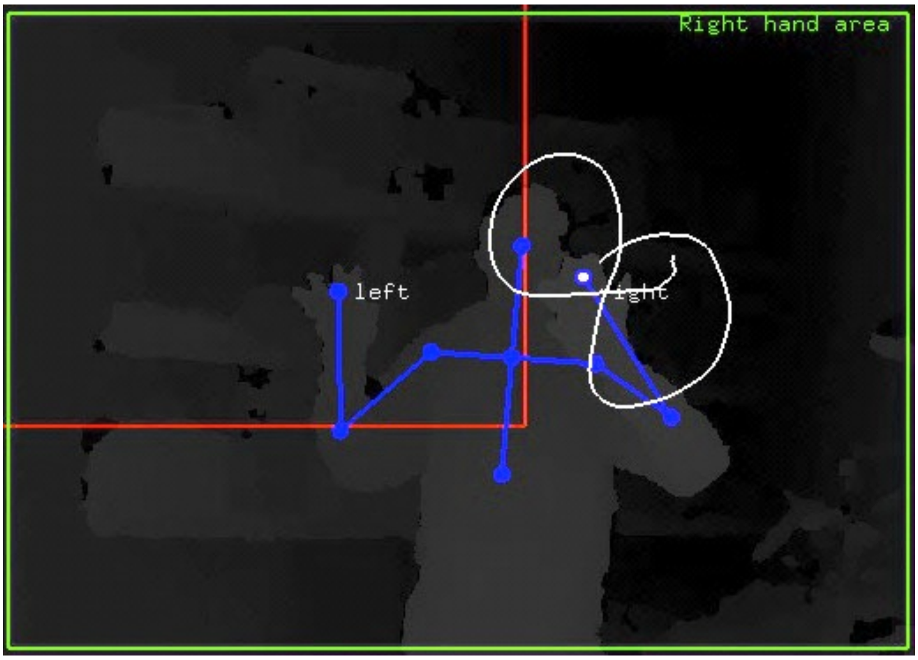


Figure 8. Hand tracking in real-time (hand path represented in white).

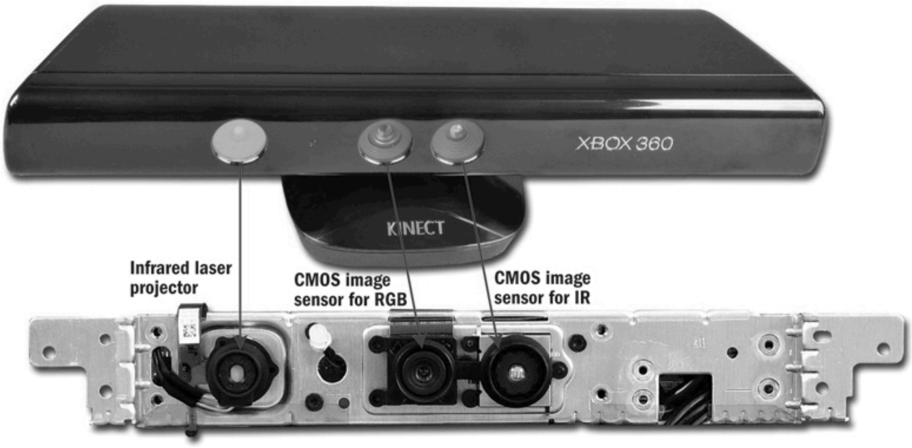


Figure 9. Inside Kinect controller [134]

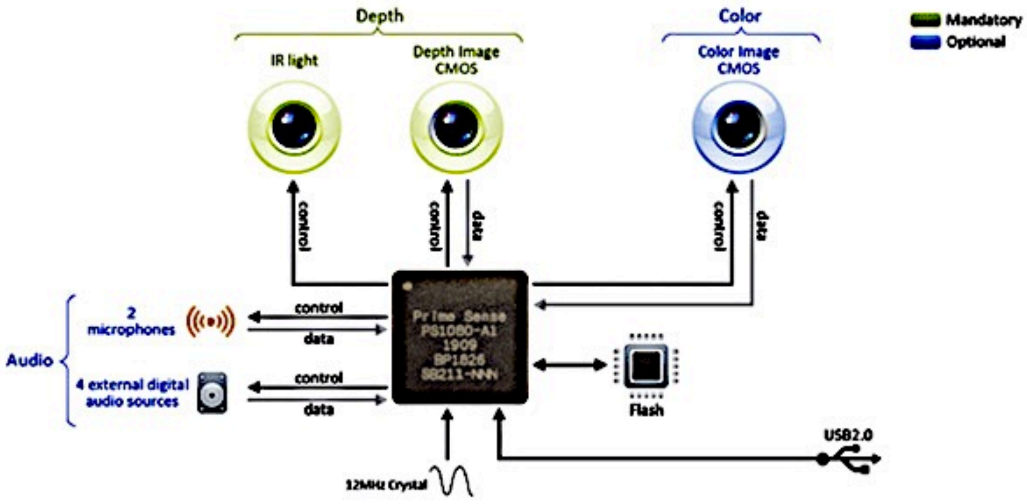


Figure 10. Kinect Interface [135]

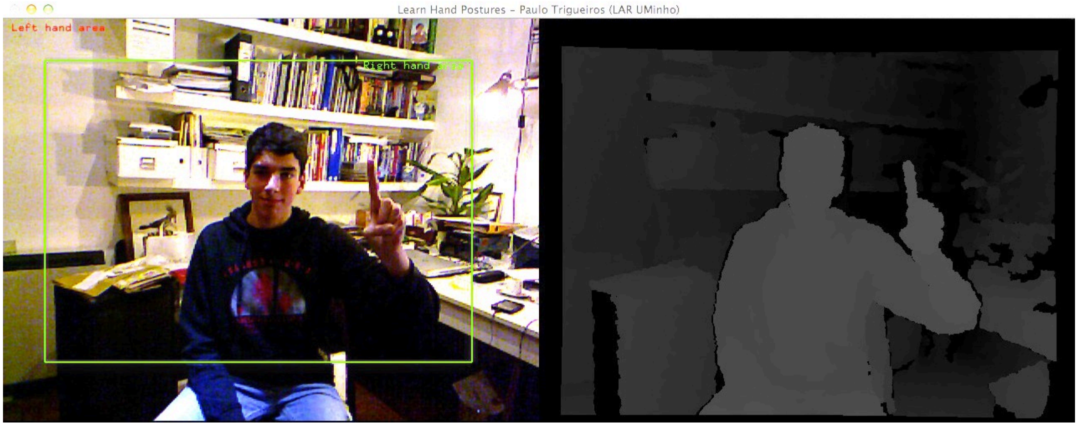


Figure 11. Kinect RGB and depth images.

Then, the hand must be segmented from the depth image, colour image or both as required, and as discussed in the following section.

3.2.2 Hand Segmentation

Hand segmentation is achieved by defining a bounding box (*handArea*) around the active hand position (*handPos*). The bounding box defines the ROI (region of interest) as shown in Figure 12, according to the following set of equations and used with the depth or grey image for segmentation:

$$distFactor = (handPos(z))/divFactor \tag{13}$$

$$handArea(x, y) = handPos(x, y) - startPos/distFactor \tag{14}$$

$$handArea.width = windowWidth/distFactor \tag{15}$$

$$handArea.height = windowHeight/distFactor \tag{16}$$

The *distFactor* value is used to keep the bounding box aspect ratio, according to the hand distance to the camera, as shown in Figure 12. The value in parenthesis represents the real depth distance.

In the current system implementation the values for *divFactor*, *startPos*, *windowWidth* and *windowHeight* were defined as 1000, 60, 120 and 110 respectively. The value for *divFactor* was obtained by experimentation, with the goal of achieving the minimum bounding box that would include the entire hand and which size would be relative to the hand distance to the camera.

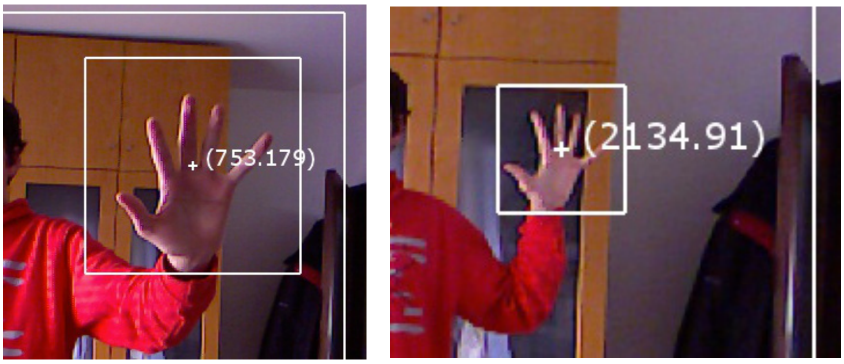


Figure 12. Bounding box size relative to the hand distance to the camera.

As soon as the bounding box is set, two parallel planes are defined, the near and far threshold planes, in the vicinity of the hand position according to the following formulas:

$$nearPlane = handPos(z) - vicinity \tag{17}$$

$$farPlane = handPos(z) + vicinity \tag{18}$$

The *vicinity* variable was defined as 200 for the current implementation. This way, a 3D bounding box is defined, as shown in Figure 13, and all the hand pixels that fall inside it are extracted, forming what we call a *hand blob* (Figure 14). The *hand blob* is a binary image, where all the pixels belonging to the hand are represented in white. The image thus obtained is used as input to the feature extraction phase.

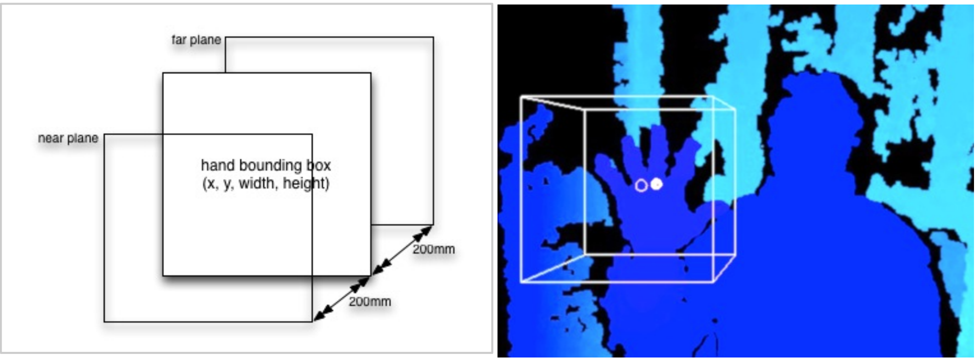


Figure 13. Hand bounding box with the near and far-threshold planes represented (left); 3D bounding box obtained for hand segmentation (right).

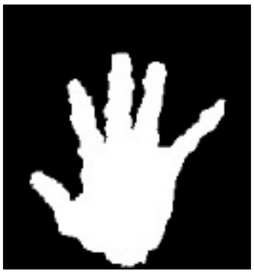


Figure 14. Hand blob extracted from the depth image and used for feature extraction.

As explained by Andersen et al. [136], one of the problems with the Kinect depth image is the presence of noise, which has implications in the quality of the final image and which means that the extracted hand position is not stable, even without apparent movement. This has implications on the quality of the final features extracted from the segmented hand, resulting in possible different values for the same hand posture, as discussed by Trigueiros et al. [8].

Trying to solve this problem, several approaches have been tried out. Camplani et al. [137], presented an efficient hole filling strategy that they claim is able to improve the quality of the depth maps. Their proposed approach is based on a joint-bilateral filtering framework that includes spatial and temporal information. The missing depth values are obtained applying iteratively a joint-bilateral filter to their neighbour pixels. Others authors like Bongalon [138] tried to solve the problem by averaging data from multiple image frames. We tested this approach, in an attempt to improve the quality of the final features, averaging over 20 frames in a first approach, but the result was a tremendous reduction on the obtained frame rate. We tried to reduce the number of frames used for averaging, but the final results were also not satisfactory. As we were using the whole depth image, the process became too heavy. A Kalman filter applied to the tracked hand position was also tested, however, the results obtained with the implemented approach were superior to the ones obtained with this method.

The final approach used, tested with good results in another implementation [6], without degrading the performance in terms of frame rate, was a cumulative average of hand position, over three frames, according to the following formulas:

$$meanPos(x,y,z) = \frac{\left(meanPos(x,y,z) * (ACCUM-1) + handPos(x,y,z)\right)}{ACCUM} \tag{19}$$

In the formula, the ACCUM variable represents the number of frames to take into consideration. Using this method, we were able to improve the precision of the extracted features with impact in the final model obtained for hand classification.

For dynamic gesture feature extraction, where the need to identify the start and end of a gesture exists, it was used a value that allows us to identify whether the hand is moving. That value is also averaged over three frames in the following manner:

$$distance = \frac{\left(distance * (ACCUM-1) + \ length(meanPos - lastPos)\right)}{ACCUM} \tag{20}$$

The *lastPos* value is updated each frame with the latest mean hand position (*meanPos*) calculate with equation 19. At system initialization the *distance* value is set to 0, and the *meanPos* and *lastPos* are initialized during the first frame acquisition with the current *handPos*.

3.3 Static Gesture Interface (SGI) module

The Static Gesture Interface module is responsible for hand feature extraction and system training for static gesture classification. Static gestures, also sometimes designated hand postures, are considered a static form of hand pose [139, 140].

For the problem of static hand gesture recognition, first it is necessary to extract meaningful features from the hand image, as explained in section 2.2.1, to train the system to recognize the required hand postures. Training implies the use of the extracted features to learn models, with the help of machine learning algorithms, that can be used in real-time human computer interaction interfaces.

This section is divided into two parts: (1) *features extraction for static hand classification* and (2) *system train and gesture classification*.

3.3.1 Feature Extraction

Careful hand features selection and extraction are very important aspects to consider in computer vision applications for hand gesture recognition and classification for real-time human-computer interaction. This step is crucial to determine in the future whether a given hand shape matches a given model, or which of the representative classes is the most similar. According to Wacs et al. [141] proper feature selection, and their combination with sophisticated learning and recognition algorithms, can affect the success or failure of any existing and future work in the field of human computer interaction using hand gestures.

Feature extraction methods determine the appropriate subspace of dimensionality m in the original feature space of dimensionality d ($m \leq d$) [76].

Thus, the problem of feature selection can be defined as follows:

Given:

1. {F(x_i), $i=1,...,d$}, a feature set of dimension d and,
2. **E**(F), the classification error

Select :

3. {S(x_j), $j=1,...,m$}, a subset of dimension m ($m \leq d$)
   such that **E**(S) is minimum

After a comparison study of different hand image features for hand pose classification [8], trying to understand the one that achieved the best results, being at the same time simple in terms of computational complexity, it was decided to use a one-dimensional function, called the *centroid distance* (section 2.2.2). This function, which is derived from the object boundary coordinates, is also called a shape signature as shown in [25, 142]. According to Zhang et al. [24] and Trigueiros et al. [8], it gives very good results in shape retrieval and classification. This type of signature can describe the shape by itself, and can be a pre-processing step for other feature extraction algorithms, like for example Fourier descriptors, as shown in [8]. Being one of a set of possible shape descriptors, the centroid distance is expressed by the distance of the contour boundary points, (x_i, y_i); $i=0,\dots, N-1$, from the object centroid (x_c, y_c), as follows:

$$d(i) = \sqrt{(x(i) - x_c)^2 + (y(i) - y_c)^2}\,, \ i = 0, \dots, N-1 \tag{21}$$

So, first it is necessary to extract the hand contour, as shown in the diagram from Figure 16 resulting in an image like the one shown in Figure 15. From the hand binary blob received as parameter, the contour is extracted and sampled to a fixed number of points (N), which in the proposed solution implementation was set to 32. This value was obtained after several tests with a number of different powers of 2, giving good final results in terms of classification accuracy. The space between two consecutive candidate points is given by the following formulae:

$$step = (size(contour)/N) + 1 \tag{22}$$

where the function *size*(*contour*) returns the number of elements in the vector that contains the hand contour points.

From the obtained discrete values, the centroid distance is calculated resulting a centroid distance feature vector. The obtained vector is saved into an instance database. Each centroid distance feature vector is calculated every 50 milliseconds, according to equation 1, and saved in an instance database similar to the one shown in Table 2.

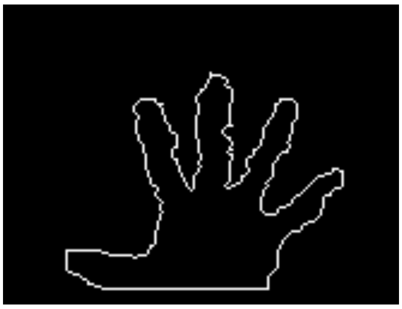


Figure 15. Hand contour extracted from hand blob.

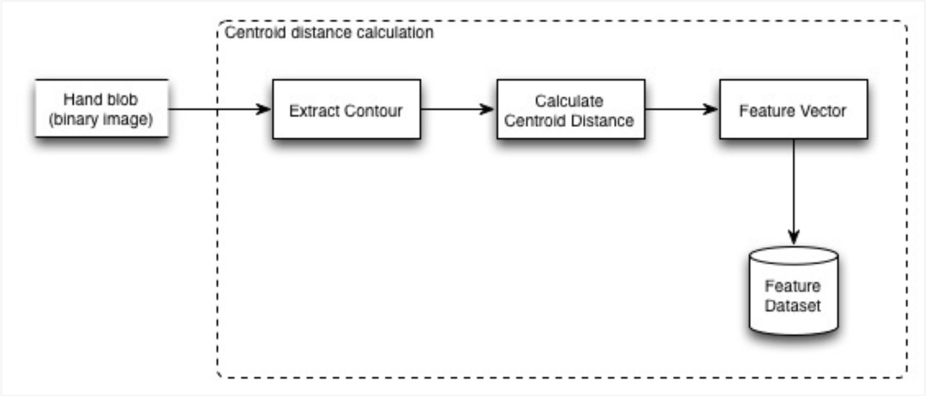


Figure 16. Centroid distance feature calculation diagram.

In terms of visual representation, each centroid distance vector can be represented graphically in the form of an histogram, called the *centroid distance signature*, as shown in Figure 17.

Table 2. Hand centroid distance feature vectors prior to normalization.

Class	Value1	Value2	Value3	Value4	Value5	…	Value31	Value32
0	39.949	34.985	37.611	34.741	34.076	...	33.014	33.377
0	39.936	34.968	37.676	34.690	34.039	...	33.011	33.350
1	48.403	43.410	37.935	31.563	28.527	...	35.943	44.043
1	48.034	42.978	37.912	31.870	28.829	...	35.775	43.601
2	55.728	45.359	31.054	19.940	25.045	...	29.461	35.766
2	55.507	45.234	31.019	20.141	25.393	...	29.943	36.426
...	...	...	...	...	...	...	...	...

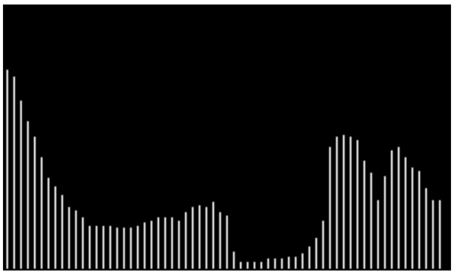


Figure 17. Centroid distance signature.

Due to the subtraction of centroid from the boundary coordinates, this operator is invariant to translation as shown by Rayi Yanu Tara et al. [19] and a rotation of the hand results in a circularly shift version of the original image. All the features vectors are normalized prior to training, by subtracting their mean and dividing by their standard deviation (*z-normalization*) [66, 143] as follows,

$$Z = (a_{ij} - \bar{a})/\sigma \quad (23)$$

where $\bar{a}$ is the mean of the instance i, and σ is the respective standard deviation, achieving this way scale invariance as desired. The vectors thus obtained have zero mean and a standard deviation of 1.

3.3.2 Train and Classification

The resulting database is used to train a multi-class Support Vector Machine (SVM) classifier and build a model capable of online hand posture classification as shown in the diagram of Figure 18. In this diagram one can see that the instances are loaded into the training system, after which they are normalized. After data normalization, the classifier is trained and the obtained model is saved.

The SVM is a pattern recognition technique in the area of supervised machine learning, which works very well with high-dimensional data. When more than two classes are present, there are several approaches that evolve around the 2-class case [144]. The one used in our system is the one-against-all approach, where *c* classifiers have to be designed and used to separate one class from the rest. One drawback with this kind of approach is that after the training phase there are regions in the space, where no training data lie, for which more than one hyper plane gives a positive value or all of them result in negative values [83].

The SVM algorithm was selected for the final implementation, because in the experiments that were carried out with the features selected in the previous section, we were able to achieve very high values of accuracy. Also, the resulting obtained model was compact and fast, able to be applied in applications with real-time classification demands.

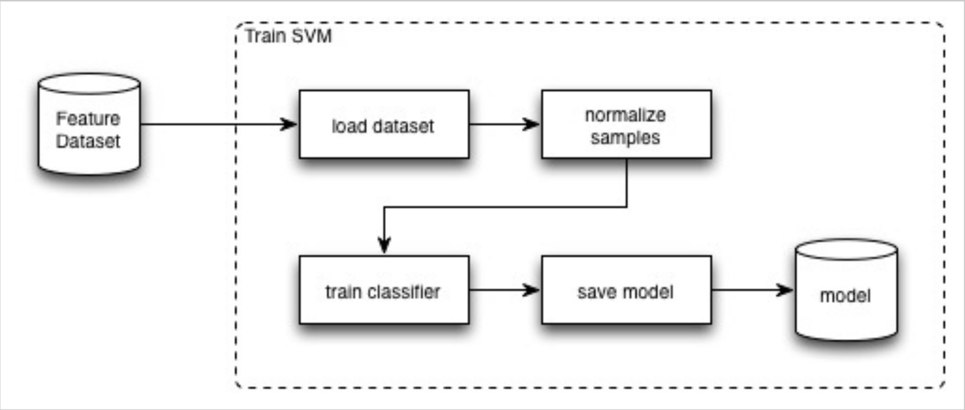


Figure 18. SVM train diagram.

So, for model training and gesture classification the open source Dlib library was used, a general-purpose cross-platform C++ library capable of SVM multiclass classification [145].

The resulting model is used during the classification process as shown in Figure 19. The user’s hand is detected and an instance feature vector is extracted. The feature vector is normalized, by the *z-normalization* method (equation 23), and the SVM model is used to predict the instance class as shown in the two examples of Figure 20 and Figure 21, where the command CLOSE, corresponding to a close hand and the command FIVE, corresponding to an open hand with all the fingers spread, are correctly classified.

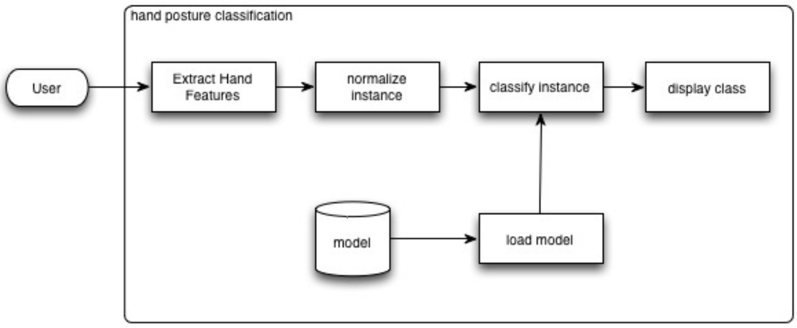


Figure 19. Hand posture classification diagram, using trained SVM model.

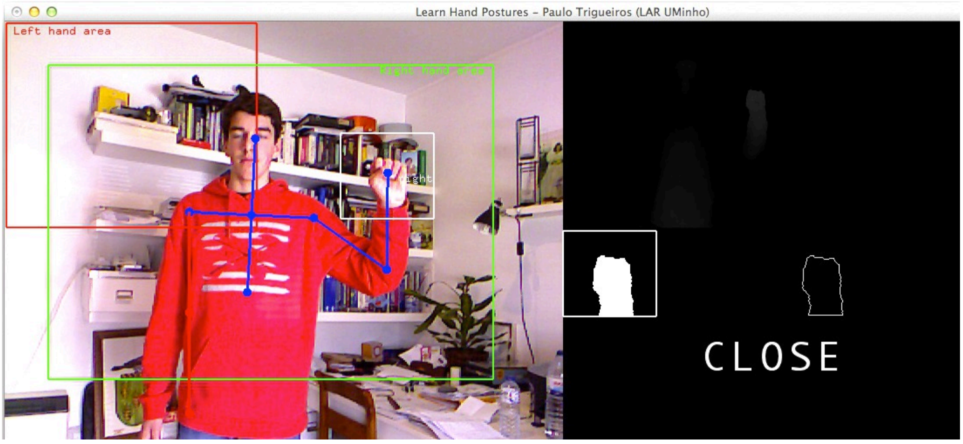


Figure 20. “Close” command detected and correctly classified.

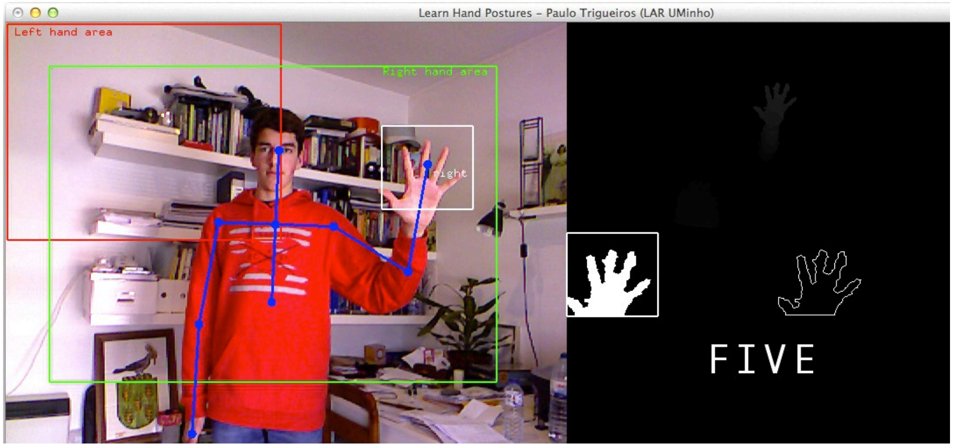
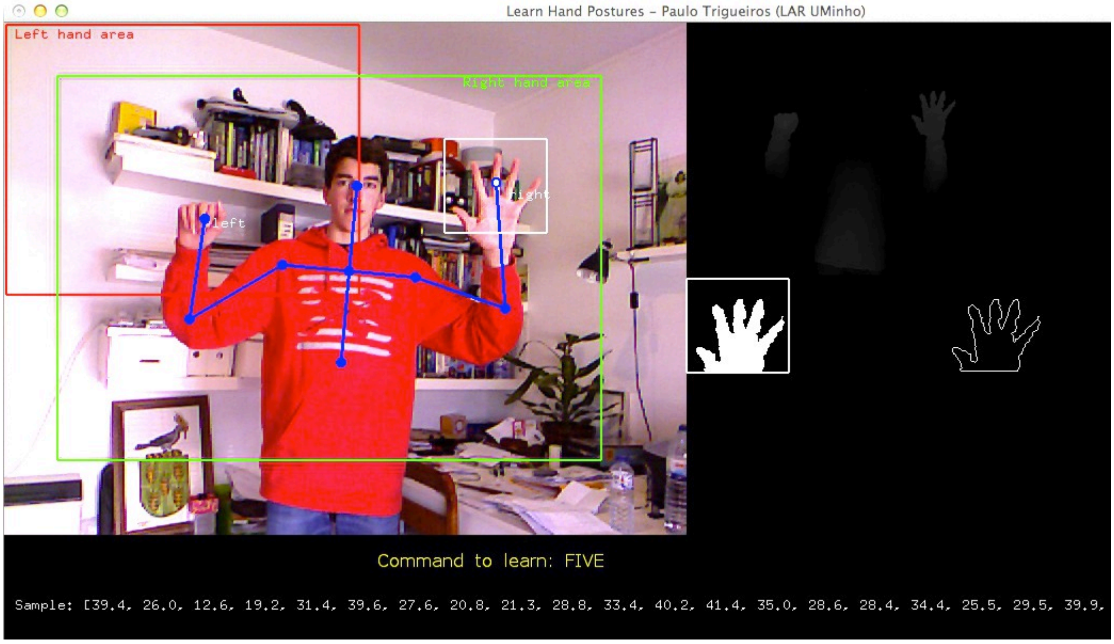


Figure 21. “Five” command detected and correctly classified.

The following image shows the user interface for hand posture model training and testing. Below the main user interface image, we have information concerning the current command being learned: name of current command and an example of an extracted instance feature vector (*sample*). The recording process is activated by raising the left hand into the red rectangle, identified in the image as the "*Left hand area*". On the left side of the interface we can see represented the extracted hand blob image and the respective hand contour used in section 3.3.1.



```
Command to learn: FIVE
Sample: [39.4, 26.0, 12.6, 19.2, 31.4, 39.6, 27.6, 20.8, 21.3, 28.8, 33.4, 40.2, 41.4, 35.0, 28.6, 28.4, 34.4, 25.5, 29.5, 39.9,
```

Figure 22. User interface for the static gesture training and testing.

3.4 Dynamic Gesture Interface (DGI) Module

The Dynamic Gesture Interface Module is responsible for hand feature extraction and model train for each one of the gestures we want the system to learn. Dynamic gestures are time-varying processes, which show statistical variations, making **HMMs** (*Hidden Markov Models*) a plausible choice for modelling the processes [146, 147]. In this way, a human gesture can be understood as a HMM where the true states of the model are hidden in the sense that they cannot be directly observed. HMMs have been widely used in a successfully way in other areas, like for example in speech recognition and hand writing recognition systems [75].

As shown in section 2.3.4, given a number of states $S = \{S_1, S_2, \dots, S_N\}$ and the number of distinct symbols in the alphabet represented as $V = \{v_1, v_2, \dots, v_M\}$, an HMM is defined as $\lambda = (A, B, \Pi)$, where:

- **A** is a matrix with all the state transition probabilities defined as:

$$\boldsymbol{A} = [a_{ij}]\ where\ a_{ij} \equiv P(q_{t+1} = S_j | q_t = S_i)\ and\ q_t\ is\ the\ state\ at\ time\ t$$

- **B** is the vector containing the observation probabilities defined as:

$$\boldsymbol{B} = \{b_j(m)\}\ where\ b_j(m) \\ \equiv P(O_t = v_m | q_t = S_j)\ and\ O\ is\ the\ observation\ sequence$$

- and Π is the initial state probabilities defined as:

$$\boldsymbol{\Pi} = [\pi_i]\ where\ \pi_i \equiv P(q_1 = S_i)$$

For a better understanding of the process taken to train a set of dynamic gestures and to learn the HMM model parameters that can be used in an online recognition system, we will divide this section into two parts: (1) *feature extraction for dynamic hand gesture classification* and (2) *system train and gesture classification*.

3.4.1 Feature Extraction

Dynamic gestures are considered in this work as the 2D path taken by hand in a certain time period, initiated with hand movement and ending when the hand stops. For the present study, the 2D features used are sufficient for the dynamic hand gesture recognition problem.

Although some authors [13, 148] define gestures as a combination of static and dynamic gestures, we have designated that in the present study as a user gesture sequence.

As shown on the diagram of Figure 23, the sequence of points extracted from the hand path, consisting of the hand centroids (x_c, y_c), is labelled according to the distance to the nearest centroid, using a clustering algorithm based on the Euclidean distance, resulting in a discrete feature vector as the one shown in Figure 24. The vector thus obtained, our observation sequence O as explained in section 2.3.4, is translated to the origin, resulting in a translation invariant feature vector as desired, and used for gesture train or classification.

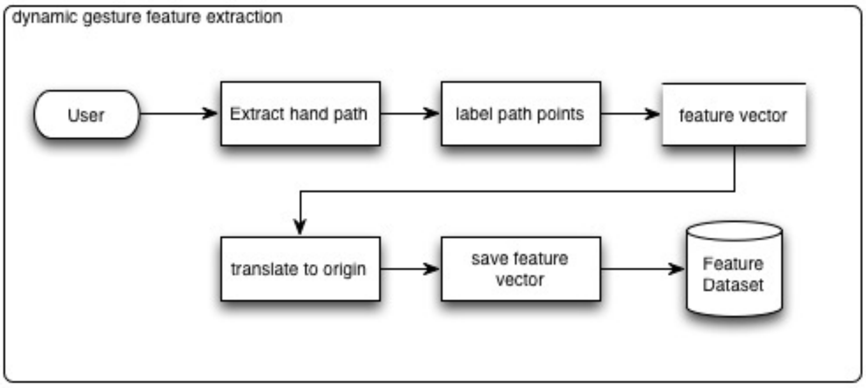
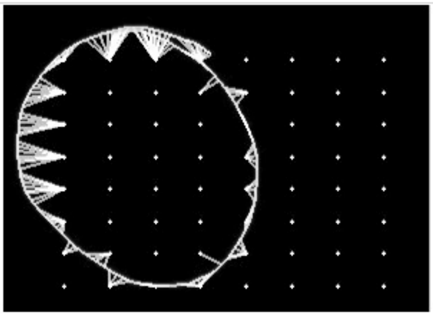


Figure 23. Dynamic gesture feature extraction diagram.



[1 2 2 2 3 3 4 4 4 5 6 13 13 14 14 14 14 14 14 14 13 13
20 19 19 18 18 17 24 23 22 22 22 22 22 29 29 29 29
29 29 29]

Figure 24. Gesture path with respective feature vector after labelling.

3.4.2 Train and Classification

The feature vectors obtained in the previous section are used to train the HMM and learn the model (λ) parameters: the initial state probability vector (Π), the state-transition probability matrix ($A=[a_{ij}]$) and the observable symbol probability matrix ($B\ [b_j(m)]$) as shown in Figure 25.

This step is one of HMMs basic problems – **the learning problem**: given a training set of observations sequences $X = \{O^k\}_k$, we want to learn the model that maximizes the probability of generating X, namely, we want to find λ* that maximizes $P(X \mid \lambda)$ as explained in [65, 66]. This is done for each gesture that we want to learn.

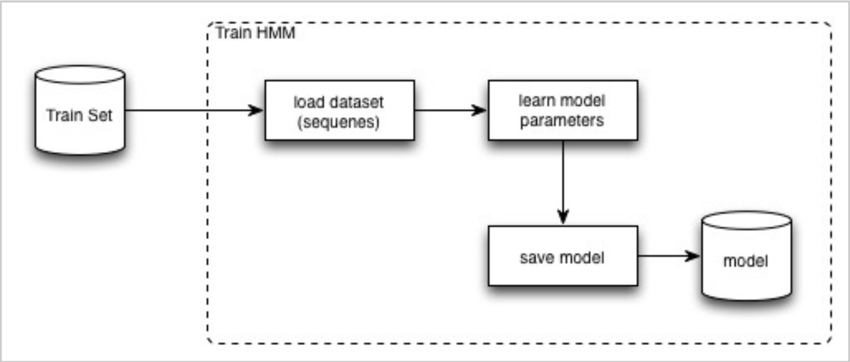


Figure 25. Learn HMM model parameters from a training set of observations.

During the recognition phase, an output score for the sample gesture is calculated for each one of the learned HMM models $\lambda=(A,B,\Pi)$, as shown in the diagram of Figure 26. The model with the highest score represents the given gesture.

This is another one of HMMs basic problems – **the likelihood problem**: given a model λ, we want to evaluate the probability of a given observation sequence, $O=\{O_1, O_2, \ldots, O_T\}$, namely, $P(O \mid \lambda)$ as explained in [65, 66].

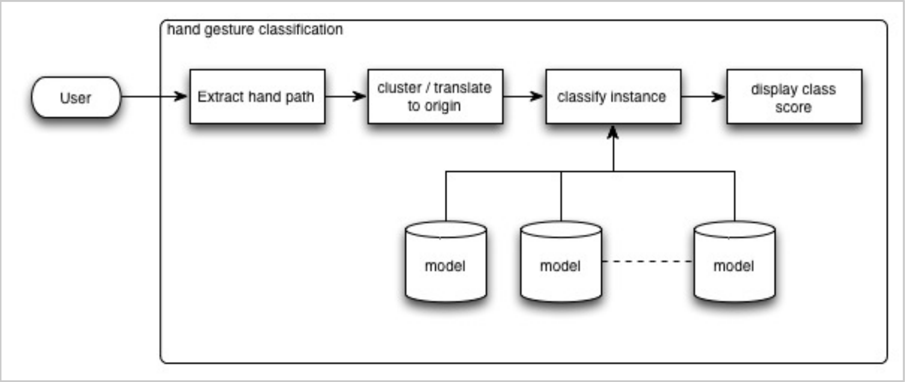


Figure 26. Output the model with the highest score based on the recognized gesture.

The following two images show examples of gestures, namely “*GOAL*” and “*DROPBALL*”, correctly classified.

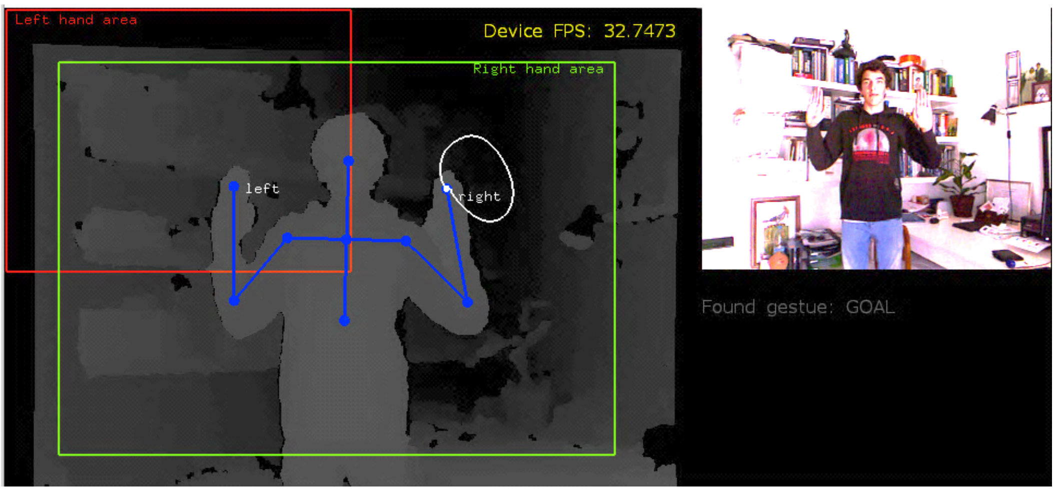


Figure 27. "*GOAL*" command correctly identified.

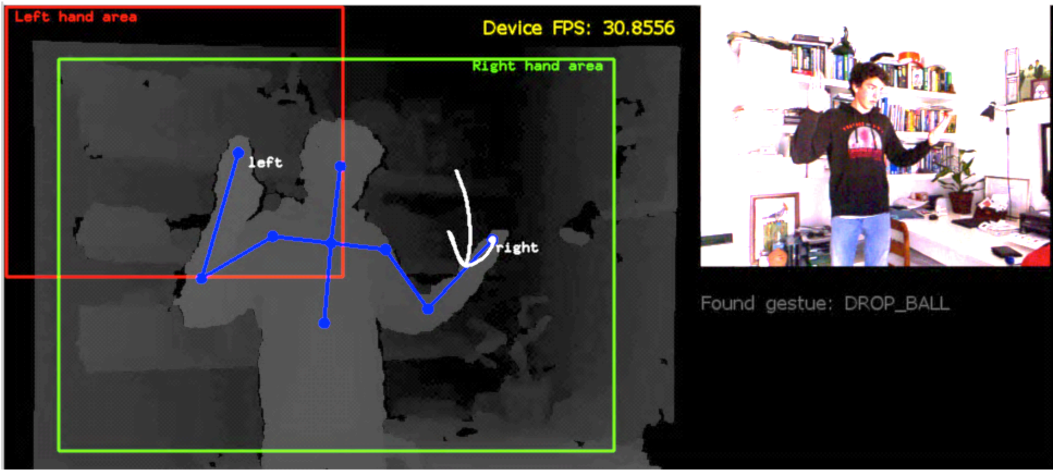


Figure 28. "*DROP-BALL*" command correctly identified.

The proposed model solution uses a left-right HMM like the one shown in Figure 29. This kind of HMM has the states ordered in time so that as time increases, the state index increases or stays de same [65, 66]. This topology has been chosen, since it is perfectly suitable to model the kind of temporal gestures present in the system as discussed in [9].

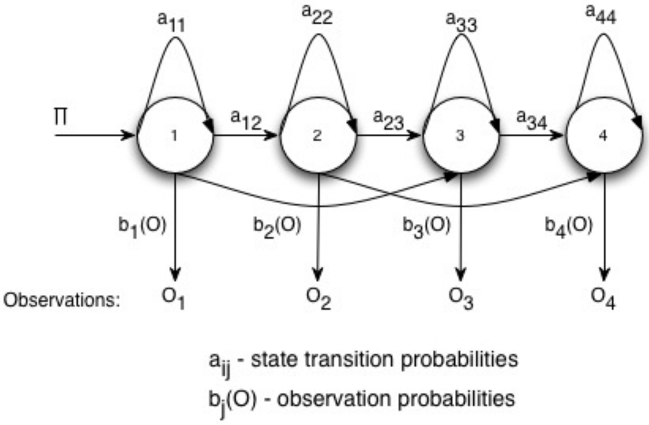
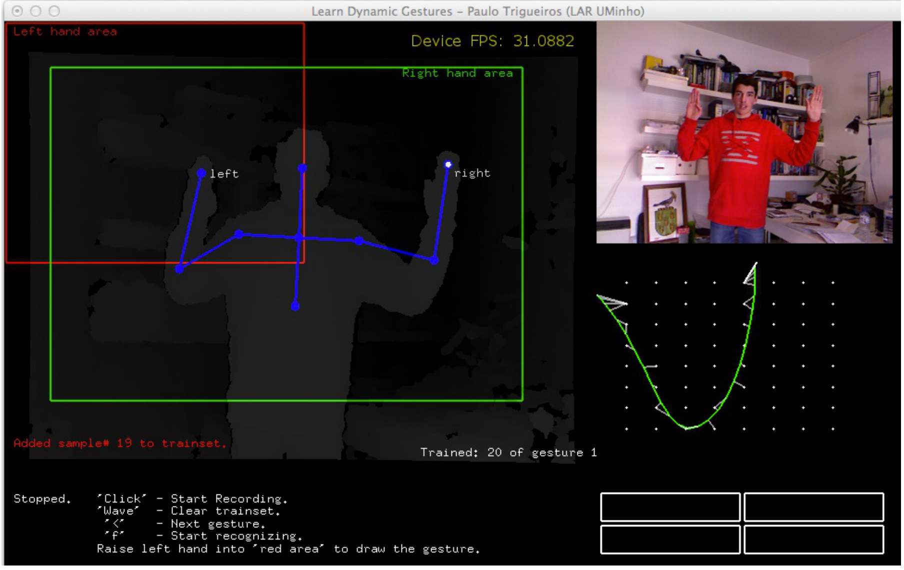


Figure 29. A 4-state left-right HMM model.

The following image shows the user interface for the dynamic hand gesture model learning and testing, where it is possible to observe, below the RGB camera image, a gesture example drawn on top of the centroids with the respective Euclidean distance lines represented in white.



```
Added sample# 19 to trainset.
Trained: 20 of gesture 1
Stopped.   'Click'  - Start Recording.
           'Wave'   - Clear trainset.
            '<'     - Next gesture.
            'f'     - Start recognizing.
           Raise left hand into 'red area' to draw the gesture.
```

Figure 30. Dynamic gestures learning interface.

3.5 Vision-based Hand Gesture Recognition System Architecture

As explained before, the design of any gesture recognition system essentially involves the following three aspects: (1) *data acquisition and pre-processing*; (2) *data representation or feature extraction* and (3) *classification or decision-making*.

Taking this into account, a possible solution to be used in any human-computer interaction system is represented in the diagram of Figure 31. The system is generic enough, and can be easily implemented using the previous described modules. For that, it uses the learned models for online human gesture recognition and classification. As can be seen in the diagram, the system has also a gesture sequence module that is responsible for building a sequence of hand gestures (static and dynamic), classify the built sequence, and transmit a command to any system interface that can be used to control a robot/system.

As explained in section 3.1, the proposed system is designed to use only one camera and is based on a set of assumptions. As it can be seen in the diagram, the system first detects and tracks the user hand, segments the hand from the video image and extracts the necessary hand features. The features thus obtained are used to identify the user gesture, using the previous learned models and loaded during system initialization. If a static gesture is being identified, the obtained features are first normalized and the obtained instance vector is then used for classification. On the other hand, if a dynamic gesture is being classified, the obtained hand path is first labelled according to the predefined alphabet, giving a discrete vector of labels which is then translated to the origin and finally used for classification. Each detected gesture is used as input into a module that builds the command sequence, i.e. accumulates each received gesture until a predefined sequence defined in the *Command Language* is found. The sequence thus obtained is classified into one of a set of predefined number of commands that can be transmitted to the GSI (generic system interface) for robot / system control.

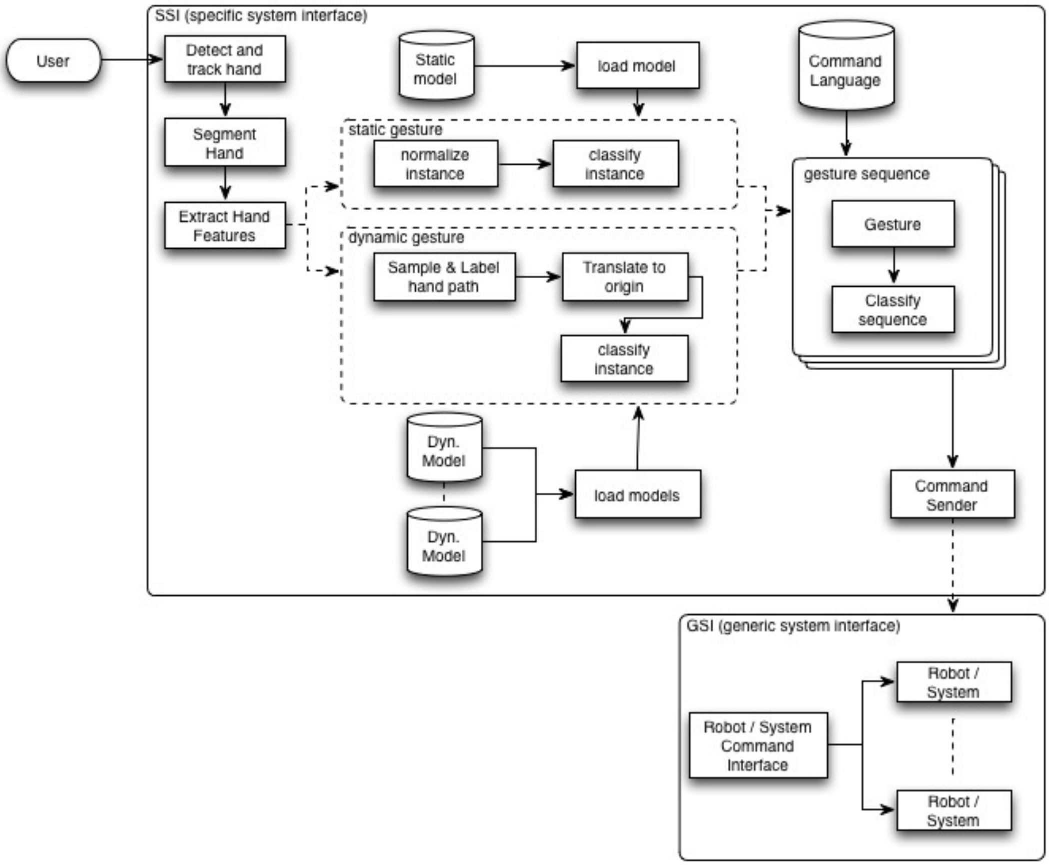


Figure 31. Vision-based hand gesture recognition system architecture.

3.6 Summary

This chapter discussed the importance of developing efficient and able to do real-time gesture recognition applications, in order to achieve human-computer interaction systems that are more intuitive and user-friendly. We saw that this type of systems have a wide range of possible applications like virtual reality, robotics and telepresence, desktop and tablet PC applications, games and sign language recognition. They also must satisfy a number of requirements in order to be successfully implemented as robustness, computationally efficiency, error tolerance and scalability. The need to have easily configurable systems was addressed, in order to ensure the necessary flexibility and scalability. Also, since the proposed solution is based on a single camera a set of assumptions that the system must obey were also described.

A proposal for a vision-based hand gesture learning system architecture, composed of four modules, has been described. The first one, the module for hand detection and tracking, addressed the problem of depth image noise and a simple but efficient solution was proposed. In the second and the third modules the static and the dynamic gesture learning interfaces were described. Finally, an integrated solution based on the discussed architecture, was proposed and described. It was shown that this type of solutions can and should be generic enough to be integrated in any human-computer interface for any robot / machine control.

Chapter 4

4 System Implementation

4.1 Introduction

The following sections discuss the algorithms and techniques implemented to solve the specific tasks necessary to build the modules described in chapter 3. Section 4.2 discusses and presents the pre-processing and hand segmentation implementations. Section 4.3 will address the static gesture module implementation. In section 4.4 the dynamic gesture module is described and in section 4.5 the integrated vision-based hand gesture recognition system implementation is discussed.

4.2 Pre-processing and hand segmentation

As explained in the previous chapter, the first step needed in any hand segmentation system is hand detection and tracking. For that, the OpenNI [131] framework, which is able to return the 3D hand position in real-time, was used. Hand segmentation is implemented in Algorithm 5. The algorithm first updates the sensor depth image information, and extracts the current hand position, from which a cumulative average position is calculated as explained in section 3.2.2. If the obtained value is inside the visible hand area and if the user is being tracked then the hand region is extracted with the *getHandImage*() function, otherwise the hand region image is cleared. The distance travelled by the tracked hand (*distance*) is also calculated. This value allows us to determine whether or not the hand is moving. The *boolean* variable *isTracking*, controls if the user is being tracked or not, by the system. The constant ACCUM represents the number of frames to take into consideration as explained in section 3.2.2.

Algorithm 5. Hand Segmentation

```
handSegmentation(), handROI
outputs:
  handROI: binary image with the extracted hand region of interest
begin
  updateCameraInformation()

  if userVisible() then
```

```
        handPos ← getHandPosition()
        if distance = 0 then
            meanPos ← handPos
            lastPos ← handPos
        endif

        meanPos ← (meanPos * (ACCUM - 1) + handPos) / ACCUM
        distance ← (distance * (ACCUM – 1) + length(meanPos – lastPos)) / ACCUM

        if getHandInside(meanPos) then
            if isTracking then
                handROI ← getHandImage(meanPos)
            else
                handROI ← Null
            endif
        endif

        lastPos ← meanPos
    endif

    return handROI
end
```

For hand segmentation, the user's hand must be inside the visible area, defined at start-up with the following values: min_x= 50, min_y= 50, max_x= 510 and max_y= 360. This viewport is called the *active hand tracking area*. The following algorithm implementation checks if the user hand is within the defined viewport.

Algorithm 6. Hand Inside Computation

```
HandInside(handPos), inside
inputs:
    handPos: a 3D point that represents the current hand position
    handviewPort: defined at start-up with $min_x$, $max_x$, $min_y$, $max_y$
outputs:
    inside: variable that represents if the hand position is inside the defined viewport

begin
    if handPos.x > handviewPort.getMinX() ^
        handPos.y > handviewPort.getMinY() ^
        handPos.x < handviewPort.getMaxX() ^
        handPos.y < handviewPort.getMaxY() then
        inside ← true
    else
        inside ← false
    endif
    return inside
end
```

To address the problem of hand segmentation, the application must update the depth pixels information and calculate the hand bounding box, as explained in section 3.2.2. As seen before, the *hand blob* is updated with all the depth image pixels that fall inside the calculated bounding box and between the two depth planes, *nearThreshold* and *farThreshold.* The hand segmentation implementation is given in the following algorithm.

Algorithm 7. Get Hand Image

```
getHandImage(handPos), binaryImage
inputs:
  handPos: the current world hand position
outputs:
  binaryImage: the binary image with the extracted hand blob

begin
  depthPixels ← getDepthPixels()
  distFactor ← $handPos_z$ / divFactor

  handArea ← $handPos_{x,y}$ - (startPos / distFactor)
  $handArea_{width}$ ← windowWidth /distFactor
  $handArea_{height}$ ← windowHeight / distFactor

  nearThreshold ← $handPos_z$ - vicinity
  farThreshold ← $handPos_z$ + vicinity

  numPixels ← $depthPixels_{width}$ * $depthPixels_{height}$
  tempImage ← 0
  for i = 0 to numPixels - 1 do
    if (depthPixels[i] ≤ farThreshold) ^ (depthPixels[i] ≥ nearThreshold) then
      tempImage[i] ← 1
    else
      tempImage[i] ← 0
    endif
  endfor

  binaryImage ← tempImage(handArea)
  return binaryImage
end
```

4.3 Static Gesture Interface Module

In the static gesture interface module, the hand features for posture classification are extracted from the segmented hand blob, obtained with Algorithm 5. The features thus obtained are used to build a dataset that is fed into the SVM training algorithm.

The decision function (model) obtained can then be used in real-time hand posture classification systems. The following algorithm implements the hand posture learn and classification technique.

Algorithm 8. Hand posture learn and classification algorithm

```
definitions:
  features: vector that contains the extracted hand centroid distances
  samples: matrix where each row is an instance of features
  labels: vector that contains for each samples row the corresponding class
  elapsedTime: represents the elapsed time for feature extraction

begin
  reset(elapsedTime)
  timeToUpdate ← 50ms

  while extractingHandFeatures do
    updateCameraInformation()

    if isTrackingUser() then
      handSegmentation()              // implemented in Algorithm 5

      if isLearning ^ (elapsedTime > timeToUpdate) then
        features = extractHandCentroid()
        updateSamples(features)
        updateLabels(currentClass)
        reset(elapsedTime)
      endif

      if isClassifying ^ (elapsedTime > timeToUpdate) then
        class = predictPostureClass(features)
        reset(elapsedTime)
        if (class ≥  minClassNumber) ^ (class ≤  maxClassNumber) then
          buildCommandString(class)
        else
          return null
        endif
      endif

    endif
  endwhile
end
```

With the proposed implementation, is possible to switch between two modes of use:

- **learning -** process for learning from features
- **recognizing** - test online the obtained model

While tracking the user, if the system is in *learning mode*, the extracted features are stored in the SAMPLES matrix, that has one row for each extracted centroid distance signature instance, and the respective LABELS vector is updated with the corresponding class being learned. If the system is in *recognizing mode*, the extracted features are used to predict the hand posture class (Algorithm 11.), and if the predicted class is within the predefined number of classes a command string, indicative of the corresponding gesture, is built.

4.3.1 Feature extraction

As explained in section 3.3.1, the features used for posture classification were obtained from the centroid distance. Algorithm 9 implements the centroid distance calculation technique. The first step is to extract the hand contour from the hand blob. For that, the OpenCV function *findContours*() is used with the contour retrieval mode set to external contour (CV_RETR_EXTERNAL), and the contour approximation mode set to store all the contour points (CV_CHAIN_APPROX_NONE).

In the proposed implementation, the number of histogram bins (*nrBins*), or the number of equally spaced points extracted from the hand contour, was set to 32. This value gives us a good compromise between number of features and recognition ability with the chosen operator. For all the given contour points, it calculates the distance from the hand centroid and saves the value in the *instance* variable. The vector thus obtained, which represents the centroid distance signature, is returned.

Algorithm 9. Centroid Distance Computation

```
centroidDistance(handImage, centroid), instance
inputs:
  handImage: image that contains the hand blob
  centroid: the point representing the hand centroid
outputs:
  instance: a vector containing the centroid signature

begin
  if (contour ← findContours(handImage)) then
    instance ← null
    nrBins ← 32
    STEP ← size(contour) / nrBins + 1
```

```

        for i = 0 to size(contour) - 1 step STEP do
            point ← contour[i]
            d[i] ← (point.x – centroid.x)^2 + (point.y – centroid.y)^2
            instance.pushback( sqrt(d[i]) )
        endfor

        if size(instance) < nrBins then
            for i = 0 to nrBins – size(instance) - 1 do
                instance.pushback(0)
            endfor
        endif

        return instance
    endif

    return null
end
```

4.3.2 Support Vector Machine (SVM) Model Training

After dataset construction the model must be trained. As explained in section 3.3.2, for model training the open source Dlib library [145] was used. For training, the *trainClassifier* algorithm implementation (Algorithm 10) uses three library functions: *randomize_samples()*, *normalize_samples()* and *train()*.

The first one, *randomize_samples*(), changes the order of the samples in the database in a random way. The second one, *normalize_samples*(), normalizes all the samples by subtracting their mean and dividing by their standard deviation (*z-normalization*). This is an important step for numerical stability, preventing one large feature from smothering others and giving us scale invariance (section 3.3.1). The last one, *train*(), is used to obtain the decision function (*model*) learned with the labelled data. The *model* thus obtained is saved for future instance classification.

Algorithm 10. Train Static Gesture Classifier

```
trainClassifier(samples, labels), model
inputs:
    samples: matrix where each row is an instance of type features
    labels: vector that contains for each 'samples' row the corresponding class

begin
    randomize_samples(samples, labels)
    normalize_samples(samples)
    model ← train(samples, labels)
```

```
  saveModel(model)
  isTrained ← true
  return model
end
```

4.3.3 Hand Posture Classification

After model training, the system is able to classify new hand postures. Algorithm 11 implements hand posture classification. It receives as input a new instance (*sample*) for classification, normalizes it, and uses the previous obtained model for class prediction.

Algorithm 11. Predict Hand Posture Class

```
predictClass(sample), class
inputs:
  sample: vector containing hand features for classification with trained model
outputs:
  class: the value for the predicted class

begin
  samp ← normalize(sample)
  class ← model(samp)
  return class
end
```

4.4 Dynamic Gesture Interface Module

The Dynamic Gesture Interface module is responsible for dynamic gesture model learning and classification. Here, the sequence of points extracted from the hand path consisting of the hand centroid mean positions calculated in Algorithm 5, are added to a gesture instance. The instances thus obtained are used to build a dataset that is fed into the HMM training algorithm in order to learn the model $\lambda = (A,B,\Pi)$ parameters. The HMM models obtained, one for each gesture, can then be used in real-time dynamic hand gesture classification systems.

The following algorithm describes the dynamic gesture learn and classification technique, where like in the static gesture module, we can switch between two modes of use:

- **learning -** process for learning from features
- **recognizing** - test online the obtained models

While tracking the user, if the system is in a *learning mode* and the hand stopped moving (*distance* < *minDistance* as explained in section 3.2.2), the extracted path is added to the TRAINSET dataset. The sample is only accepted as valid if it has more than 10 points in size. The dataset is built with one record for each hand path sample. On the other hand, if the system is in a *recognizing mode*, the extracted hand path is passed as an argument to the *classifyGesture* function (Algorithm 17.) in order to predict the corresponding gesture class. If a class is found, a command string indicative of the gesture is built.

Algorithm 12. Dynamic gesture learn and classification algorithm

```
definitions:
  sample: vector that contains the hand path points
  distance: controls the distance travelled by the user hand
  minDistance: threshold to control if the hand is moving or not
  found: the gesture found

begin
  while extractingHandFeatures() do
    updateCameraInformation ()

    if isTrackingUser() then
      handSegmentation()  // implemented in Algorithm 5
      if isLearning then
        if distance < minDistance then
          addPathToGestureDataset()
          gestureReset()
        else
          handMoving ← true
          addSample(meanPos)
        endif
      endif

      if isRecognizing then
        if distance < minDistance then
          found ← classifyGesture()
          if found then
            buildCommandString(found)
          endif
          gestureReset()
        endif
      else
        addSample(meanPos)
      endif

    endif
    endwhile
end
```

4.4.1 Feature Extraction

In this phase, as explained in section 3.4.1, the hand path sequence of points must be labelled according to the minimum distance to a set of predefined centroids, using the Euclidean distance metric. Algorithm 14 implements this procedure and returns a discrete feature vector containing the labels that are added to the "*trainset*" dataset. The obtained dataset will then be used for learning the HMM model parameters.

The following algorithm implements the add gesture technique, which is called every time a new user gesture is detected. It receives as parameter a set of points that represent the hand path, verifies if the size of the sample vector is valid and calls the *toObservation* function for gesture labelling.

Algorithm 13. Add gesture points to Trainset

```
addGesture(sample)
inputs:
  sample: the set of points that are part of the gesture

begin
  minSampleSize ← 10
  if size(sample) > minSampleSize then
    labels ← toObservation(sample)          // implemented in Algorithm 14
    trainset.add(labels)
  endif
end
```

The *toObservation* algorithm first rescales the set of gesture points to the predefined hand viewport, the hand visible area as explained in section 4.2, and then labels all the sample points according to the defined alphabet (section 2.3.4).

Algorithm 14. Label sample points according to defined alphabet

```
toObservation(points), labels
inputs:
  points: the set of points to be labelled according to the defined alphabet
outputs:
  labels: discrete vector containing the labelled points (observation)

begin
  rescale(points)                           // implemented in Algorithm 15
```

```
    N ← size(points)

    for i = 0 to N - 1 do
        label ← labelPoint(points[i])          // implemented in Algorithm 16
        labels.add(label)
    endfor

    return labels
end
```

Algorithm 15. implements the rescale procedure. The min_element() function finds the smallest elements in the range (*first*, *last*) where *first* is the first element in the vector passed as argument and *last* is the last element in the vector. The **max_element**() function finds the greatest element in the range (*first*, *last*). The function **map**(*value*, *inputMin*, *inputMax*, *outputMin*, *outputMax*), re-maps a given number from one range to another. In other words, it converts the *value* parameter, where $inputMin < value < inputMax$, into a number where $outputMin < value < outputMax$.

Algorithm 15. Rescale points to follow inside predefined hand viewport

```
rescale(points, window)
inputs:
    points: the hand path points
    window: the predefined hand viewport (hand visible area)

begin
    $min_x$ ← 0
    $min_y$ ← 0
    $max_x$ ← 0
    $max_y$ ← 0

    if size(points) > 1 then
        $min_x$ ← min_element(points.x)
        $min_y$ ← min_element(points.y)
        $max_x$ ← max_element(points.x)
        $max_y$ ← max_element(points.y)
    endif

    w ← $max_x – min_x$
    h ← $max_y – min_y$
    if w > h then
        $target_w$ ← $window_{width}$
        ratio ← $target_w$ / w
        $target_h$ ← h * ratio
    else
        $target_h \leftarrow window_{height}$
        $ratio \leftarrow target_h / h$
        $target_w \leftarrow w * ratio$
    endif

    $sizePoints \leftarrow$ size(points)
    for i = 0 to sizePoints - 1 do
        $points[i].x \leftarrow$ map(points[i].x, $min_x$, $max_x$, 0, $target_w$)
        $points[i].y \leftarrow$ map(points[i].y, $min_y$, $max_y$, 0, $target_h$)
    endfor
end
```

Gesture labelling is an important step for gesture learning and classification. It is in this phase that the output symbols are obtained, as explained in section 2.3.4. Algorithm 16. implements the solution for this problem, i.e., it labels a given point according to the minimum Euclidean distance to the set of centroids. The algorithm receives as parameters the point to label and the centroids (*alphabet*) and returns a label for the given point.

Algorithm 16. Label point according to defined alphabet

```
labelPoint(point, centroids), label
inputs:
    point: the point to be labelled using the Euclidean distance metric
    centroids: the set of output symbols (alphabet) defined
outputs:
    label: label assigned to the point

begin
    k ← size(centroids)
    for j = 0 to k - 1 do
        distance ← (point.x – centroids.x[j])^2 + (point.y – centroids.y[j])^2

        if j = 0 then
            minDistance ← distance
            label ←  centroids[j]
        elseif distance < minDistance then
            minDistance ← distance
            label ←  centroids[j]
        endif
    endfor

    return label
end
```

4.4.2 Learn HMM Parameters.

In this section, the "*trainset*" from section 4.4.1 is used to learn all the model parameters. For that each dataset record is first converted into a proper library row vector and passed as a parameter to the *train* function.

4.4.3 Gesture Classification

After model training, the obtained classifier can be used in new gesture classifications. Algorithm 17. implements the gesture classification for a given hand path sample. If the sample size is invalid or too small the function just returns a null value, otherwise the sample is passed to the classification function and the obtained gesture class is returned.

Algorithm 17. Gesture classification for a given hand path

```
classifyGesture(sample), class
inputs:
   sample: the set of points that constitute the hand gesture
outputs:
  class: the value for the predicted gesture class

begin
  minSampleSize ← 80
  if size(sample) < minSampleSize then
    return null
  else
    class ← classify(sample)
    return class
  endif
end
```

4.5 Vision-based Hand Gesture Recognition System

The Vision-based Hand Gesture Recognition System is a system that tracks the user's hands using a single depth camera, and is able to recognize dynamic and static gestures for human/computer interaction. The system is also able to build a combination of dynamic and static gestures as commands that can be used for remote robot/system control. For this, there is a need to model the command semantics. A Finite State Machine (FSM) is a usually employed technique to handle this situation [64, 149]. In the system, the FSM shown in the diagram of Figure 32 and in the state transition table (Table 3) was implemented to control the transition between the three possible defined states: DYNAMIC, STATIC and PAUSE. A *state transition table*, as the name implies, is a table that describes all the conditions and the states those conditions lead to. In the proposed solution a PAUSE state is used, giving the possibility to identify transitions between gestures and somehow eliminate all unintentional actions between DYNAMIC/STATIC or STATIC/STATIC gestures.

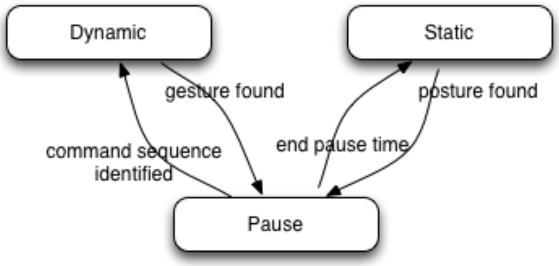


Figure 32. The Referee CommLang finite state machine diagram.

Table 3. The Referee CommLang state transition table.

Current State	**Condition**	**Sate transition**
Dynamic	Found gesture	Pause
Static	Found posture	Pause
Pause	End pause time	Static
Pause	Command sequence identified	Dynamic

Algorithm 18 implements the proposed solution for the vision-based hand gesture recognition system. When the system starts tracking the user, it switches between the three possible states. A sequence of dynamic and static gestures can be modelled as possible commands, which can then be used in any robot/control system interface.

During user tracking, the current right hand position is retrieved and the system tests if the left hand is inside the left hand viewport (control viewport), which activates gesture recognition. If during gesture recognition before a gesture is recognized the user removes its hand outside the control viewport, the gesture information is cleared.

If the finite state machine is in a DYNAMIC state then a particular dynamic gesture is classified whenever the tracked hand has stopped moving, controlled by the *minDistance* variable, and the gesture is only considered valid if the distance travelled by the hand has a certain size, as explained in section 4.4. Whenever a dynamic gesture is detected, the state machine information is updated with the gesture number and the corresponding gesture name and the gesture information are cleared. If on the other hand, the finite state machine is in a STATIC state, the tracked hand is segmented and features are extracted. The features are used to predict the hand posture class with the model obtained in section 4.3.2, and if the predicted class is within the predefined number of classes, the state machine information is updated with the gesture number and the corresponding gesture name. The PAUSE state is entered every time a gesture or hand posture is found, and exited after a predefined period of time or when a command sequence is identified, as can be seen in the state transition table (Table 3).

Algorithm 18. Vision-based hand gesture recognition system

```
definitions:
  handPos: the current world hand position
  state: the state machine current state (DYNAMIC or STATIC)
  isTracking: boolean value that controls if the user's hand is being tracked
  distance: controls the distance travelled by the user hand
  minDistance: threshold to control if the hand is moving or not
  numberOfClasses: the total number of gesture classes defined

begin
  while trackingUser() do
    updateCameraInformation()

    if isUsingSkeleton then
      handPos ← getHandPosition()
      isTracking ← getHandInside()  // implemented in Algorithm 6.

      meanHandPos ← calculateMeanHandPosition(handPos)
      calculateDistanceTravelled()

      if state = DYNAMIC then
        if isTracking then
          if distance < minDistance then
            found ← classifyGesture()
            if found then
              updateStateMachine(state)
              gesture ← getGestureName(found)
              changeStateMachine(PAUSE)
            endif

            gestureReset()
          else
            addHandPath(meanHandPos)
          endif
        else
          gestureReset()
        endif
      else if state = STATIC then
        if handInsideViewport() then
          if isTracking then
            handSegmentation()                    //----- implemented in Algorithm 5
            features ← extractHandFeatures()
            class ← predictHandPostureClass(features)

            if class ≥ 0 ^ class ≤ numberOfClasses then
              updateStateMachine(state)
              gesture ← getGestureName(class)
              changeStateMachine(PAUSE)
            endif

          endif
        endif
      endif
    endif

  lastHandPos ← meanHandPos
  endwhile
end
```

4.6 Summary

In this chapter, the fundamental algorithms that are part of the Vision-based Hand Gesture Recognition System have been described. Section 4.2 discussed and presented the pre-processing and hand segmentation implementations. The need for the user's hand to be within the visible area, *the active hand tracking area*, was once again here reinforced. Section 4.3 addressed the static gesture module implementation. For model training, the number of features used was 32, which as discussed, gives a good compromise between the number of features and the recognition ability. It was seen that in this module, the user can switch between two modes of use: *learning* and *recognizing*, giving the user the possibility to test the just trained model. Section 4.4 discussed and has given implementations for the dynamic gesture module. For hand gesture classification it was seen that if the hand path is too small, then the gesture is considered invalid and a null value is returned. Here, as in the static gesture module, the user can switch between to modes of use: *learning* and *recognizing*. Finally in the last section the integrated vision-based hand gesture recognition system implementation was discussed, where the need to model the command semantics was spoken and a solution based on a finite state machine was proposed. It was seen that using this solution, the system could switch between three possible states: DYNAMIC, STATIC and PAUSE. As discussed, the PAUSE state was introduced here, as a possible solution to model the still difficult problem that is: identify the start and end of a gesture, or gesture transition.

Chapter 5

5 Case Studies

5.1 Introduction

During the course of the study, with the aim of achieving the objectives that were proposed in chapter 1, a set of questions arose that gave rise to several applications and prototypes.

One of the first issues faced, was the problem of detecting and tracking the hand. When it was decided to start using the Kinect it was possible to explore some of its technologies in order to try to answer that issue. From those experiments it was implemented an application that gave a user the possibility of remotely control a robot in terms of direction and speed of movement with a set of simple hand commands. For these experiments an MSL robot [150] from the Laboratory of Automation and Robotics at the University of Minho [151] was used, which allowed testing the effectiveness of the proposed solution and the final implementation and which gave rise to the final prototype called *Vision-based Remote Hand Robot Control.*

In a second stage it was necessary to start identifying hand features, which could be used with machine learning algorithms in a supervised way for teaching a computer / robot understand a set of human gestures. During that study, some tests conducted by other authors were found in order to be able to detect fingertips as possible hand features. It was decided to test those features in order to verify whether they could be used robustly in real-time systems for human-robot interaction. From those experiments came a prototype called *Vision-based Hand Robotic Wheel Chair Control*, that allowed a user to drive a robotic base wheelchair, developed at Laboratory of Automation and Robotics in the University of Minho [152], through a finite number of finger commands. The experiments were carried out with the aid of a MSL robot soccer player, since the base used on those robots is equal to the chair base on which the system was intended to be deployed.

Following the study and with the purpose of testing the proposed solutions, developed to date, in a system capable of interpreting static and dynamic gestures, an application was implemented capable of recognizing a set of commands defined in a new formal language for the MSL referee. Since the Laboratory of Automation and Robotics participates actively on the RoboCup national and international robotic soccer competitions, an idea arose of trying to find a solution based on vision, capable of human gestures recognition, that could be used to solve some of the problems that were identified and that happen during competitions. Thus, emerged an application called ReCLIS (*Referee Command Language Interface System*), capable of interpreting in real-time a set of referee commands, send them to the RefereeBox (referee's assisting technology) that transmits them to the robots.

In order to test the validity of the hand features being used and compare them with a new idea that arose for possible hand features, and at the same time test the validity of the proposed solutions in other types of applications, another prototype was developed, still in testing phase, able to interpret Portuguese Sign Language.

The following sections describe in detail each one of the applications and prototypes implemented as well as some of the algorithms used.

5.2 Vision-based Remote Hand Robot Control

5.2.1 Introduction

The Vision-based Remote Hand Robot Control is an application that enables a user to remotely control a robot with a number of simple hand commands committed in front of a Kinect camera. The Kinect camera is used to gather depth information and extract the user’s hand in order to use that to calculate control information to transmit to the robot. The depth image is used to detect and track the hand by the nearest point approach as explained in the following section (Hand Segmentation). After hand detection, two planes are defined as minimum and maximum thresholds for hand segmentation. The extracted hand blob is used to calculate the hand centroid (relative position), direction of turning, direction of movement, and the linear velocity that are transmitted to the robot.

The system is composed mainly of two modules:

1. The data acquisition and pre-processing module
2. The *processing module* where all the values used to be transmitted to the robot, are calculated.

The following image shows the diagram of the proposed system, where the various modules are represented.

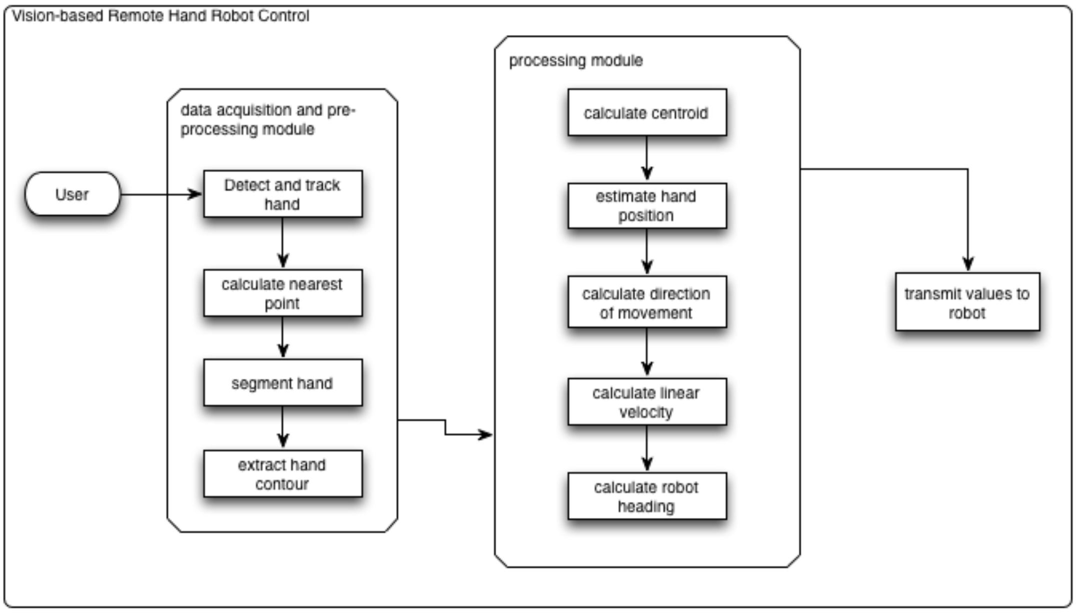


Figure 33. Vision-based Remote Robot Control diagram.

5.2.2 Hand Segmentation

In order to segment the hand region and extract the hand blob, the user's nearest point to the camera is first calculated according to equation 24. For that, the current depth image is passed as a parameter to the *calculateNearestPoint* algorithm (Algorithm 19). For each time *t*, the closest point on the depth image *I* is calculated according to the formula:

$$minDistance = min\big(I(x,y) : 0 \leq x \leq height(I) \;\&\; 0 \leq y \leq width(I)\big) \qquad (24)$$

where *I(x,y)* is the current depth image and *height*(I) and *width*(I) are the respective image width and height values.

Using this *minDistance* value, two parallel planes in the Z-direction are defined with values equal to *minDistance - margin* and *minDistance +margin*, in order to extract the hand blob and the hand contour. The constant value *margin*, added and subtracted from the *minDistance,* was defined as 5 which is sufficient to cover all the depth pixels of interest.

The following algorithm implements the nearest point calculation, giving the depth image and its size, and returns a point that represents the minimum distance to the camera and the resulting segmented hand binary blob.

Algorithm 19. Nearest point computation

```
calculateNearestPoint(depthImage, width, height), minDistance, thImage
inputs:
  depthImage: current frame depth image
  width: depth image width
  height: depth image height
outputs:
  minDistance: point representing the user minimum distance to the camera
  thImage: the resulting binary mask

begin
  nrPixels ← width * height
  margin ← 5

  minDistance ← depthImage [0]
  for i = 1 to nrPixels - 1 do
    if minDistance = 0 ^ depthImage [i] > 0 then
      minDistance ← depthImage [i]
    else if depthImage [i] > 0 ^ depthImage [i] < minDistance then
      minDistance ← depthImage [i]
    endif
  endfor

  nearThreshold ← minDistance - margin
  farThreshold ← minDistance + margin

  for i = 0 to nrPixels - 1 do
    if (depthImage[i] > nearThreshold) ^ (depthImage[i] < farThreshold) then
      thImage[i] ← 1
    else
      thImage[i] ← 0
    endif
  endfor

  return minDistance, thImage
end
```

The hand centroid is calculated according to equation 25 and equation 26. The value thus obtained is used as the relative hand position.

$$\bar{x} = \frac{1}{n}\sum x_i \tag{25}$$

$$\bar{y} = \frac{1}{n}\sum y_i \tag{26}$$

Where x_i and y_i are the coordinates of the pixels that belong to the hand blob (white pixels), and *n* is the number of pixels in the blob.

A new value for the hand position is estimated with a Kalman filter [153] [154], thereby enabling a smoother hand path, and respectively a smoother robot movement. This smoother robot movement not only makes it more visibly attractive but also increases the life expectancy of the robot motors.

The robot direction of movement is calculated according to equation 27, given the hand vector angle related to the image centre as illustrated in Figure 34.

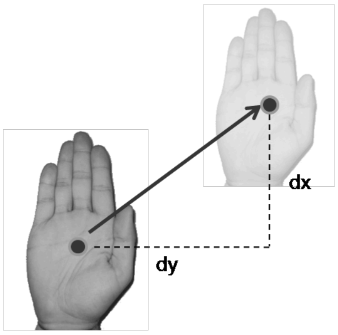


Figure 34. Robot direction of movement relative to hand position.

$$direction = atan2(dx, dy) \tag{27}$$

The vector length represented by the distance between the image centre and the hand centroid, is then used to calculate the robot linear velocity, which is proportional to that value according to equation 28.

$$linearV = \sqrt{dx^2 + dy^2}/divFactor \tag{28}$$

Where *linearV* is the linear velocity transmitted to the robot and the *divFactor* was defined as 6, introduced here in order to avoid sudden accelerations. The *divFactor* value was adjusted during some experiments.

5.2.3 Orientation

Hand orientation is calculated taking into account two vectors:

1. the vector $\vec{a}$ formed between the hand centroid and the farthest point from it.
2. the vector $\vec{b}$ parallel to a horizontal line that passes through the centre of the image as shown in Figure 35.

The angle θ is then obtained by using the dot product between the two vectors according to equation 29.

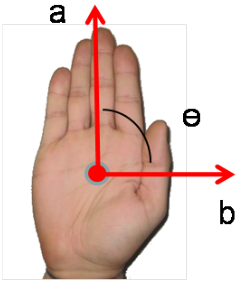


Figure 35. Vectors used for robot heading calculation.

$$\theta = arcos((a \cdot b)/\|a\|\|b\|\ ) \tag{29}$$

Being $a \cdot b$ the dot product between the two vectors and $\|a\|$ the norm of the vector. The angle θ is used to control robot heading (left or right), as shown in Figure 36.

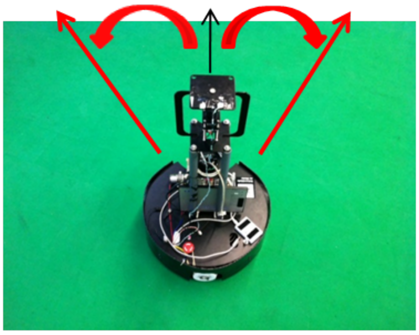


Figure 36. Robot heading dependent on hand rotation.

The following algorithm calculates the angle theta and orientation (left or right), given the hand contour, extracted in section 5.2.2, and the camera image centre.

Algorithm 20. Angle and hand orientation computation

```
calculateOrientation(handContour, imageCenter, width, height), theta, orientation, vAngular
inputs:
  handContour: the current hand contour extracted from the hand blob image
  imageCenter : centre of camera image
  width: depth image width
  height: depth image height
outputs:
  theta : the hand angle used to control the robot heading
  orientation : the indicative text of turning (left, ritght)
  vAngular: the angular velocity to be transmitted to the robot

begin
  distance ← 0
  nrPoints ← getContourNrPoints()

  for i = 0 to nrPoints - 1 do
    if $handContour[i]_y$ < $imageCenter_y$ then
      v ← ($handContour[i]_x$ - $imageCenter_x$, $handContour[i]_y$ - $imageCenter_y$)
    endif
    lenght ← $sqrt(v_x^2 + v_y^2)$
    if distance < length then
      distance ← length
      maxPoint ← ($handContour[i]_x$, $handContour[i]_y$)
    endif
  endfor

  a ← ($maxPoint_x$ - $imageCenter_x$, $maxPoint_y$ - $imageCenter_y$)
  b ← (width - $imageCenter_x$, 0)
  $length_a$ ← $\mathbf{sqrt}(a_x^2 + a_y^2)$
  $length_b$ ← $sqrt(b_x^2 + b_y^2)$

  //------------ normalize vectors ------------
  a ← a / $length_a$
  b ← b / $length_b$
  theta ← arcos((a · b) / ||a||||b||)

  rightAngleTh ← 80
  leftAngleTh ← 100
  angularVelocity ← 5

  if theta < rightAngleTh then
    orientation ← 'Turn right'
    vAngular ← - angularVelocity
  elseif theta > leftAngleTh then
```

```
            orientation ← 'Turn left'
            vAngular ← angularVelocity
        else
            orientation = 'Forward'
            vAngular ← 0
        endif

        return theta, orientation, vAngular
    end
```

5.2.4 Prototype implementation

In order to validate the prototype with the proposed method, a MSL (Middle Size League) soccer robot from Minho team was used to carry out a series of experiments. A computer connected to a Kinect camera grabs hand movements and communicates the calculated information (heading, angular velocity and linear velocity) through Wi-Fi to the robot on-board computer, as it attempts to show the image form Figure 37. The robot motion speed transmitted to the robot is proportional to the vector length that connects the hand centroid to the camera image centre point (red vector in Figure 38), and hand orientation gives us robot-heading direction (blue line in Figure 38).

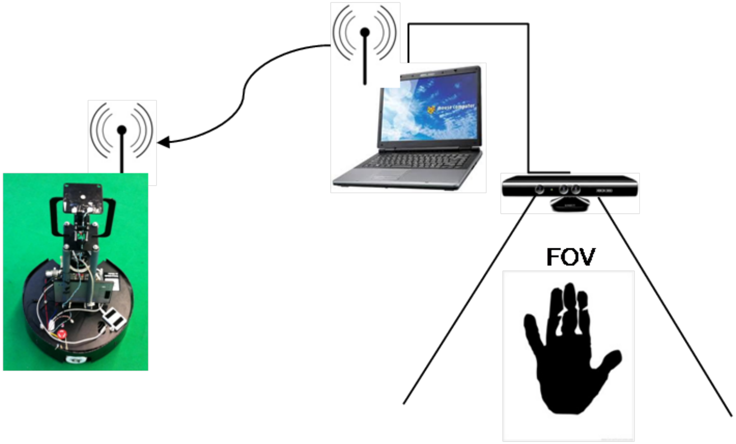


Figure 37. Computer connected to a Kinect camera for remote hand gesture robot control.

The human-computer interface for the prototype (Figure 38) was developed using the C++ language and the *openFrameworks* toolkit with the OpenCV [28] and the *libfreenect* addons under Ubuntu. OpenCV was used for the vision-based operations, like hand contour extraction, and *libfreenect* was used to control the Kinect camera and get the RGB and depth images.

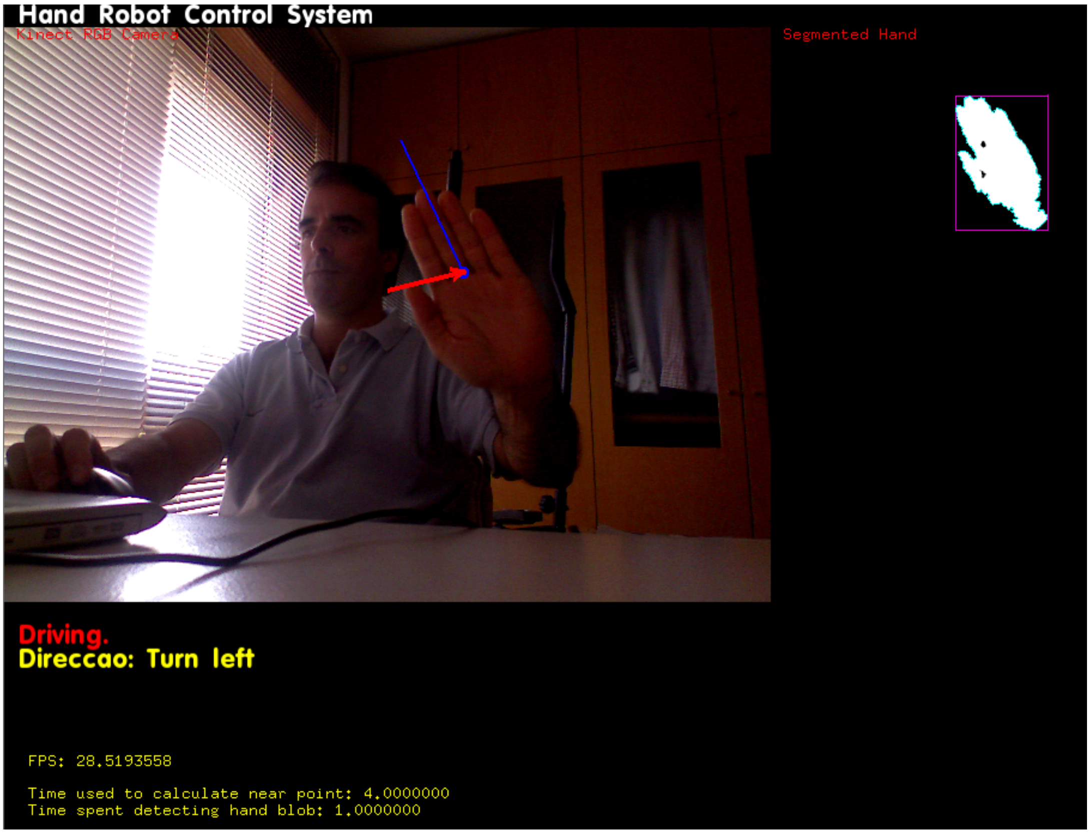
The system main algorithm implementation is shown below. Every frame the depth image is updated and the hand blob and corresponding contour are extracted. From the obtained hand blob, the hand centroid is extracted, for direction of movement computation, and the hand orientation is calculated (Algorithm 20.). The values for the linear velocity, angular velocity and direction of turning must be updated every frame and transmitted to the robot.

Algorithm 21. *Vision-based Remote Robot Control*

```
definitions:
  nrBlobs: number of blobs found in the depth image
  blob: the extracted hand blob
  vLinear:  the linear velocity to be transmitted to the robot
  direction: the direction of movement to be transmitted to the robot

begin
  while trackingUser() do
    updateSensorData()
    width ← getImageWidth()
    height ← hetImageHeight()
    depth ← getCurrentFrameDepthImage()
    calculateNearestPoint(depth, width, height)          //---- implemented in Algorithm 19
    estimateHandPosition()

    blob ← segmentHand()
    nrBlobs ← extractHandBlobContours()

    $imageCenter_x$ ← 320
    $imageCenter_y$ ← 240

    if nrBlobs > 0 then
      centroid ← calculateHandCentroid(blob)
      calculateOrientation(blob, centroid, width, height) //---- implemented in Algorithm 20.
      dy ← $centroid_x$ - $imageCenter_x$
      dx ← $imageCenter_y$ – $centroid_y$

      direction ← atan2(dy, dx) * 180/PI
      if direction < 0 then
        direction ← 360 + direction
      endif

      vLinear ← sqrt(dx^2 + dy^2) / 6
      trasmitToRobot(vLinear, vAngular, direction)
```

```
        endif

    endwhile
end
```

The tests made with the implemented solution showed that this system is able to operate in real-time, taking 4_{ms} to calculate the near point for hand segmentation, and around 1_{ms} to extract the hand blob as can be seen in the information area in the HCI interface on Figure 38. The communication with the robot is carried out through Wi-Fi, and transmission speed is dependent on the network conditions.



```
Driving.
Direccao: Turn left

FPS: 28.5193558

Time used to calculate near point: 4.0000000
Time spent detecting hand blob: 1.0000000
```

Figure 38. Human computer interface for the Vision-based Remote Hand Robot Control.

5.3 Vision-based Hand Robotic Wheelchair Control

5.3.1 Introduction

The Vision-based Hand Robotic Wheelchair Control is an application that enables a user to drive a robotic base wheelchair with a minimum number of finger commands made in front of a Kinect camera. The diagram of Figure 39 shows the modules that compose the proposed system and the flow of information.

The system is composed of two main modules:

1. a data acquisition and pre-processing module.
2. a *processing module*, where the finger peaks are extracted to calculate the values that are going to be transmitted to the robotic platform to control the wheel-chair.

The application uses a Kinect camera to gather depth information, segment the user's hand and extract useful information, enabling the calculation of control information that is transmitted to the wheelchair robotic base. The depth image is used to detect the hand by the nearest point approach as explained in section 5.3.3.

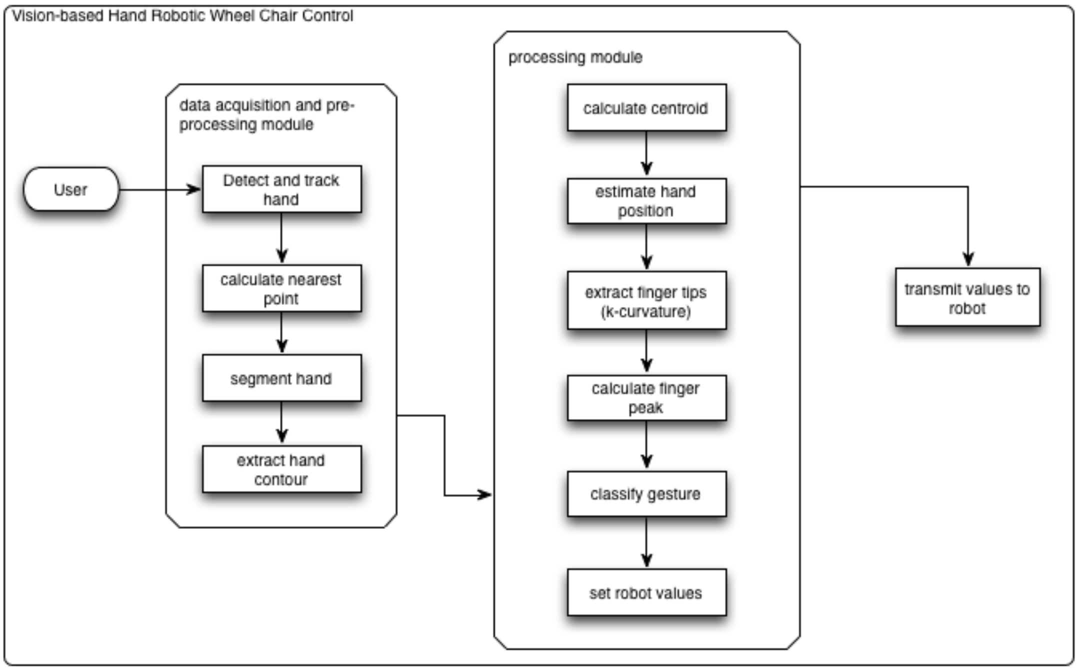


Figure 39. Vision-based Hand Robotic Wheel Chair Control diagram.

After having detected the hand position, two planes are defined as minimum and maximum thresholds for hand segmentation. From the obtained hand blob, the hand contour is extracted, and the number of fingertips and their position on the hand contour are calculated using the *k-curvature algorithm* [155], as explained in section 5.3.4.

The index finger is used to control the forward movement and direction of turning (left or right) as shown in Figure 40 (a, b, c). The thumb controls the lateral displacement (left or right) as shown in Figure 40 (e, f). Backward movement is carried out with two fingers forming a 'V' structure (Figure 40 d), and the closed hand is used as the stop command. After finger peak extraction, the gesture is classified and the respective command information is transmitted to the robotic platform.

The main goal of the proposed system consists of giving the user the capability to control a robotic-based wheelchair without the need to touch any physical devices. With this kind of technology, we expect that people with disabilities can gain a degree of independence in performing daily life activities, being at the same time an alternative to some of the already existing solutions.

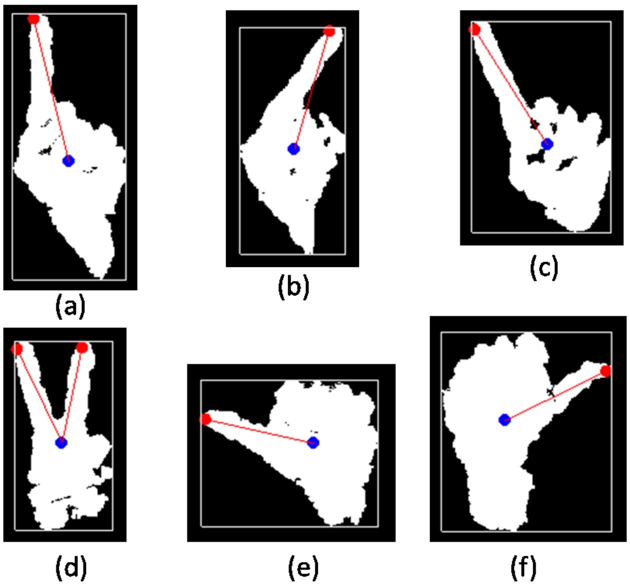


Figure 40. Finger commands used to drive the wheelchair.

5.3.2 The Wheelchair Command Language Definition

In order to implement all the finger commands this system accepts, a new and formal language definition was created: the *WheelChair CommLang*. As in [156], the language must represent all the possible gestures and at the same time be simple in its syntax. The language was defined using BNF (Bakus Normal Form or Bakus-Naur Form) [157]:

- Terminal symbols (keywords and operator symbols) are in a CONSTANT-WIDTH TYPEFACE.
- Choices are separated by vertical bars (|) and in greater-than and less-than symbols (< CHOICE>).
- Optional elements are in square brackets ([optional]).
- Sets of values are in curly braces ({SET}).
- A syntax description is introduced with ::=.

The language has only one type of command: the *Drive Command*. The DRIVE_COMMAND is composed of a COMMAND and a SPEED value. For the COMMAND the following options are available: STOP, MOVE_FORWARD, TURN_RIGHT, TURN_LEFF, MOVE_RIGHT and MOVE_LEFT commands. The SPEED value controls the velocity that we can transmit to the robotic base in the following set of values: HIGH, MIDDLE, LOW.

```
<LANGUAGE> ::= {<DRIVE_COMMAND>}

<DRIVE_COMMAND> ::= <COMMAND> [<SPEED>]
SPEED::= <HIGH> | <MIDDLE> | <LOW>
COMMAND ::= <STOP> | <MOVE_FORWARD> | <TURN_RIGHT> | <TURN_LEFT> |
                     <MOVE_RIGHT> | <MOVE_LEFT>


<STOP> ::= STOP
<MOVE_FORWARD> ::= MOVE_FORWARD
<TURN_RIGHT> ::= TURN_RIGHT
<TURN_LEFT> ::= TURN_LEFT
<MOVE_RIGHT> ::= MOVE_RIGHT
<MOVE_LEFT> ::= MOVE_LEFT
```

5.3.3 Hand Segmentation

In order to segment the hand region, the nearest point to the camera is calculated on each frame. For each time t, the closest point on the depth image I is calculated according to equation 24.

Using the obtained *minDistance* value, two parallel planes (*minDistance*-5, *minDistance*+5) are defined to extract the hand blob from which the contour is calculated. The hand contour is then used to detect fingertips using the *k-curvature algorithm* from the next section.

5.3.4 The k-curvature algorithm

The *k-curvature* is an algorithm that attempts to find pixels that represent peaks and valleys along the contour of an object [155], in our case the hand contour.

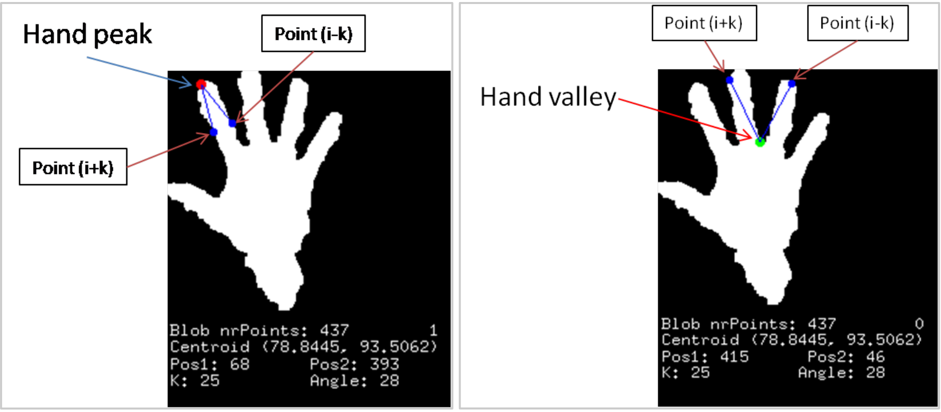


Figure 41. Hand peak and valley point detection

At each pixel position ***i*** in the hand contour ***C*** (shown as '*Hand peak*' in Figure 41), the *k-curvature* is determined by calculating the angle between the vectors A=[C_i, C_{i-k}] and B=[C_i, C_{i+k}], where *k* is a constant set equal to 30 in the prototype implementation. The angle can be easily calculated using the dot product between the two vectors (equation 29) as illustrated in Figure 42.

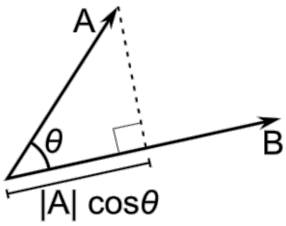


Figure 42. Dot product between vectors A and B

An angle threshold equal to 35° was used, such that only values below this angle will be considered further.

In order to classify the points as peaks or valleys, the cross product between the vectors is calculated (Figure 43). If the sign of the z component is positive the point is labelled as a peak and stored, otherwise the point is a valley and is discarded.

It was found that the hand contour used led to the detection of a set of peaks in the neighbourhood of the strongest locations as candidate peaks, so, an average of all these points on each finger is calculated in order to define one single peak per finger.

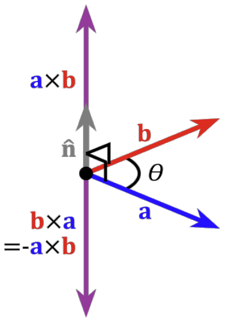


Figure 43. Cross product between two vectors

The following algorithm implements the fingertip detection on the hand contour passed as parameter and returns the number of fingertips found and their position on the hand contour.

Algorithm 22. Fingertip detection and extraction

```
detectFingerTips(handsContour), fingers, fingerPeaks
inputs:
   handContour: the current hand contour extracted from the hand blob binary image
outputs:
   fingers: the number of found finger tips
   fingerPeaks: vector containing all the finger peaks found

begin
   fingers ← 0
   fingerPeaks ← null
   fingerK ← 30
   nrPoints ← getContourNrPoints(handContour)

   for i = 0 to (nrPoints – fingerK) - 1 do
      if i < fingerK then
         v1← (handContour[i]$_x$- handContour[nrPoints + i - fingerK]$_x$,
                 handContour[i]$_y$ - handContour[nrPoints + i - fingerK]$_y$)
      else
         v1 ← (handContour[i]$_x$ - handContour[i - fingerK]$_x$,
                    $handContour[i]_y$ - $handContour[i - fingerK]_y$)
        endif
        v2 ← ($handContour[i]_x$ - $handContour[i + fingerK]_x$,
                $handContour[i]_y$ - $handContour[i + fingerK]_y$)

        if i < fingerK then
            $v1_{3D}$ ← ($handContour[i]_x$- $handContour[nrPoints + i - fingerK]_x$,
                        $handContour[i]_y$- $handContour[nrPoints + i - fingerK]_y$, 0)
        else
            $v1_{3D}$ ← ($handContour[i]_x$- $handContour[i - fingerK]_x$,
                        $handContour[i]_y$- $handContour[i - fingerK]_y$, 0)
        endif

        $v2_{3D}$ ← (handContour[i].x - handContour[i + fingerK].x,
                    handContour[i].y - handContour[i + fingerK].y, 0)

        vXv ← crossProduct($v1_{3D}$, $v2_{3D}$)
        $lenght_{v1}$ ← sqrt($v1_x^2 + v1_y^2$)
        $lenght_{v2}$ ← sqrt($v2_x^2 + v2_y^2$)

        // ------------- normalize vectors ---------------
        v1 ← v1 / $lenght_{v1}$
        v2 ← v2 / $lenght_{v2}$

        theta ← arcos(($v1 \cdot v2$) / ||v1||||v2||)

        // ------------- if theta < 35 then we are at a peak or valley ( /\ or \/ ) -----------------
        mininumAngle ← 35
        if abs(theta) < mininumAngle then
            if $vXv_z > 0$ then
                fingerPeaks.pushback(handContour[i])
                fingers ← fingers + 1
                fingerFound ← true
            endif
        elseif fingerFound = true then
            fingerFound ← false
        endif
    endfor

    if fingers > 0 then
        calculateOrientation()
        return fingers, fingerPeaks
    endif
end
```

As explained earlier, during the fingertip detection, a set of peaks in the neighbourhood of the strongest locations are detected as candidate tips so, an average of all the saved peak points on each finger is calculated to define one peak per finger. The following algorithm implements that solution.

Algorithm 23. Finger point computation

```
calculateFingerPoint (fingerPeaks), fingerBlobs
inputs:
   fingerPeaks: an array with the candidate finger peaks
outputs:
   fingerBlobs: vector containing all the finger peaks found

begin
   nrFingers ← 0
   for i = 0 to size(fingerPeaks) - 1 do
      sumX ← 0
      sumY ← 0
      nrPoints ← 0

      for j = 0 to size(fingerPeaks[i]) - 1 do
         sumX ← sumX + $fingerPeaks[i][j]_x$
         sumY ← sumY + $fingerPeaks[i][j]_y$
         nrPoints ← nrPoints + 1
      endfor

      averageX ← sumX / nrPoints
      averageY ← sumY / nrPoints
      tempPoint ← (averageX, averageY)
      fingerBlobs.pushBack(tempPoint)
      nrFingers ← nrFingers + 1
   endfor
end
```

5.3.5 Direction of turning

The wheelchair turning direction θ is calculated by the dot product between the control vector, vector between the hand centroid and the fingertip, and a horizontal vector parallel to a line that crosses the image centre as illustrated in Figure 44 and given by equation 29.

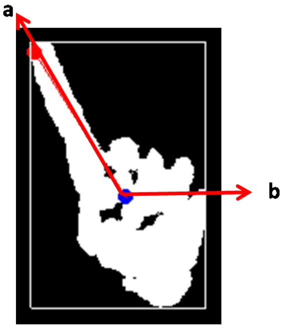


Figure 44. Vectors used in the calculation of finger orientation

Orientation calculation is given by Algorithm 24, based on the calculated finger blobs, the hand centroid and the number of fingers found. The function returns a number that represents the type of finger orientation and that is used to build the command that is sent to the robotic base. A value of zero represents the "*move forward*" command (Figure 40 a), a value equal to one represents the "*move back*" command (Figure 40 d), a value equal to two or three represents the "*turn right*" and "*turn left*" commands respectively (Figure 40 b and c), a value equal to four or five represents the "*to the right*" or "*to the left*" commands (Figure 40 e and f) and a value equal to -1 means that no fingers were found. In that case the values to be transmitted for heading, angular velocity and linear velocity, are all set to zero, meaning a stop command.

Algorithm 24. Orientation computation based on fingertips

```
calculateOrientation (fingerBlobs, centroid, nrFingers), orientation
inputs:
  fingerBlobs: the position of all the finger peaks
  centroid: the hand blob centroid coordinates
  nrFingers: the number of finger peaks found
outputs:
  orientation: number that represents the type of finger orientation

begin
  $v1 \leftarrow (fingerBlobs[1]_x - centroid_x,\ fingerBlobs[1]_y - centroid_y)$
  $v2 \leftarrow (640 - centroid_x, 0)$
  $lenght_{v1} \leftarrow \mathbf{sqrt}(v1_x^2 + v1_y^2)$
  $lenght_{v2} \leftarrow \mathbf{sqrt}(v2_x^2 + v2_y^2)$

  // ------------ normalize vectors ---------------
  $v1 \leftarrow v1 / lenght_{v1}$
  $v2 \leftarrow v2 / lenght_{v2}$
```

```
    fingerAngle ← arcos((v1 · v2) / ||v1||||v2||)

    maxRightAngle ← 90
    minRightAngle ← 70
    maxLeftAngle ← 130
    minLeftAngle ← 110
    maxToRightAngle ← 30
    minToRightAngle ← 0
    maxToLeftAngle ← 180
    minToLeftAngle ← 150

    if nrFingers = 1 then
        if fingerAngle < maxRightAngle ^ fingerAngle > minRightAngle then
            orientation ← 2
        elseif fingerAngle < minLeftAngle ^ fingerAngle > maxLeftAngle then
            orientation ← 3
        elseif  fingerAngle > minToRightAngle ^ fingerAngle < maxToRightAngle then
            orientation ← 4
        elseif  fingerAngle < maxToLeftAngle ^ fingerAngle > minToLeftAngle then
            orientation ← 5
        else
            orientation ← 0
        endif
    elseif nrFingers = 2 then
        v1← $(fingerBlobs[1]_x - centroid_x, fingerBlobs[1]_y - centroid_y)$
        v2← $(fingerBlobs[1]_x - centroid_x, fingerBlobs[1]_y - centroid_y)$
        $lenght_{v1}$ ← sqrt($v1_x^2 + v1_y^2$)
        $lenght_{v2}$ ← sqrt($v2_x^2 + v2_y^2$)

        // ------------- normalize vectors ---------------
        v1←v1/ $lenght_{v1}$
        v2 ← v2 / $lenght_{v2}$
        fingerAngle ← arcos((v1 · v2) / ||v1||||v2||)

        moveBackAngle ← 40
        if fingerAngle < moveBackAngle then
            orientation ← 1
        endif
    else
        orientation ← -1
    endif

    return orientation
end
```

5.3.6 Prototype implementation

The Human-Computer Interface (HCI) for the prototype (Figure 45) was implemented using the C++ language, and the *openFrameworks* toolkit with the

OpenCV and the Kinect addons, *ofxOpenCv and ofxKinect* respectively, under Ubuntu. OpenCV was used for some of the vision-based operations like extracting the hand blob contour, and the Kinect addon was responsible for the RGB and depth image acquisition. In the HCI interface image it is possible to see below the segmented hand, the direction of movement, all the values given for linear velocity, angular velocity, and direction. We have also the number of fingers found and the angle obtained as explained in section 5.3.4. The fingertip extraction algorithm used was a reimplementation of a non-working add-on for the *openFrameworks*, where all the modifications were given back to the community.

The proposed method consisted of a new way to control a robotic wheelchair with the use of fingertips as hand features, based on a Kinect camera system to facilitate the extraction of this information. One major advantage of the proposed method consists on its simplicity, which leads to rapid learning rates, and gives the user the needed independent mobility. Also, the use of inexpensive hardware and open source software tools makes it a solution that can easily be applied to many other applications where this type of human-computer interfaces could help improve the quality of human life. The solution is not intended to be the best or only solution, but an alternative to the many solutions that exist in the market at the moment [158]. The solution is not universal - every disabled person is different and has different needs.

The main algorithm implementation for the prototype is given in Algorithm 25. Every frame the depth image is updated and the hand blob and corresponding contour are extracted. From the obtained hand blob, fingertips and the number of fingers are calculated and the respective values and commands are set and transmitted to the robot.

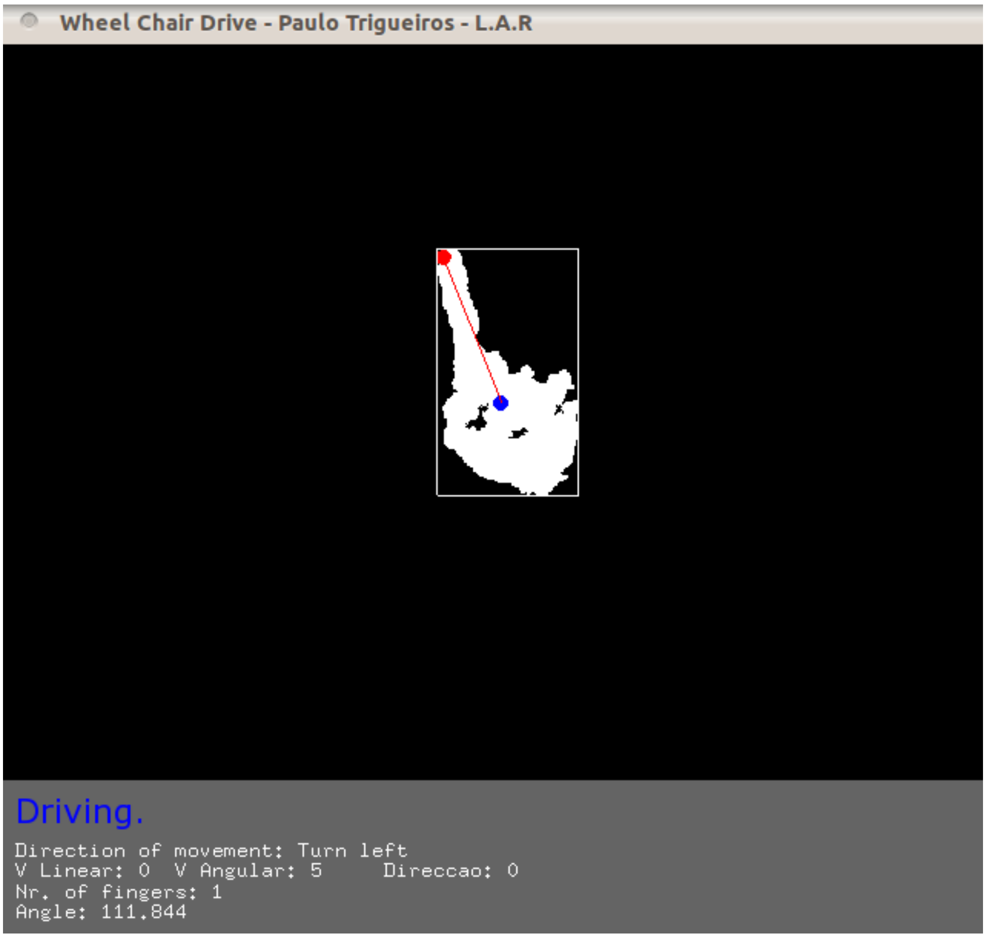
In the proposed solution, and for test purposes, two constants were defined, the linear velocity (*VLIN*) and the angular velocity (*VANG*), with values equal to 10.

Algorithm 25. Vision-based Hand Robotic Wheel Chair Control

```
definitions:
    nrBlobs: number of blobs found in the depth image
    nrFingers: number of fingers found in the hand blob
    blob: the extracted hand blob
    vLinear:  the linear velocity to be transmitted to the robot
    vAngular: the angular velocity to be transmitted to the robot
```

```
  direction: the direction of movement to be transmitted to the robot
  command:  the string command representation

begin
  VLIN ← 10
  VANG ← 10

  while trackingUser() do
    updateSensorData()
    image ← getCurrentFrameDepthImage()
    calculateNearestPoint(image)
    extractNearPoints(image)

    nrBlobs ← findDepthImageContours()
    if nrBlobs > 0 then
      blob ← getDepthImageBlob(image)
      boundingBox ← calculateBlobCentroid(blob)
      nrFingers ← findFingerTips(blob)

      if nrFingers = 0 then
        vLinear ← 0
        vAngular ← 0
        direction ← 0
        command ← 'Stop'
      elseif nrFingers = 2 then
        orientation ← getFingerOrientation()
        if orientation = 0 then
          vLinear ← VLIN
          vAngular ← 0
          direction ← 0
          command ← 'Move forward'
        elseif orientation = 1 then
          vLinear ← VLIN
          vAngular ← 0
          direction ← 180
          command ← 'Move back'
        elseif orientation = 2 then
          vLinear ← 0
          vAngular ← -VANG
          direction ← 0
          command ← 'Turn right'
        elseif orientation = 3 then
          vLinear ← 0
          vAngular ← VANG
          direction ← 0
          command ← 'Turn left'
        elseif orientation = 4 then
          vLinear ← VLIN
          vAngular ← 0
          direction ← 90
```

```
                    command ← 'To the right'
                elseif orientation = 5 then
                    vLinear ← VLIN
                    vAngular ← 0
                    direction ← 270
                    command ← 'To the left'
                else
                    command ← null
                endif
                transmitToRobot(vLinear, vAngular, direction)
            endif
        endif
    endwhile
end
```



```
Driving.
Direction of movement: Turn left
V Linear: 0  V Angular: 5     Direccao: 0
Nr. of fingers: 1
Angle: 111.844
```

Figure 45. HCI for the Vision-Based Hand Robotic Wheelchair Control prototype

5.4 Referee CommLang Prototype

5.4.1 Introduction

The *Referee CommLang Prototype* is a real-time vision-based system that is able to interpret a set of commands defined for the MSL (Middle Size League) referee, and send them directly to the *RefereeBox* [159] (referee's assisting technology), which transmits the proper commands to the robots. The commands were defined in a new formal language described in section 5.4.2 and given in Table 4.

With the proposed solution, there is the possibility of eliminating the assistant referee, thereby allowing a more natural game interface. The application uses a finite state machine, as the one described in section 4.5, for referee command construction. As stated before, the system is always in one of three possible states: DYNAMIC, STATIC and PAUSE. On start-up, the system enters the DYNAMIC state and waits until a dynamic gesture is correctly classified. When this happens, the system enters the PAUSE state for a predefined period of time, necessary to model the transitions between gestures. After that time, the system transitions to the STATIC state and remains in this state until a valid command sequence, composed of one or more hand postures, is found. At this point the system returns to the DYNAMIC state, waiting for a new command sequence.

The following sections describe the Referee Command Language (*Referee CommLang*) and the prototype implementation.

5.4.2 The Referee Command Language Definition

This section presents the *Referee CommLang* keywords with a syntax summary and description. The *Referee CommLang* is a new and formal definition of all commands that the system is able to identify. As in [156], the language must represent all the possible gesture combinations (static and dynamic) and at the same time be simple in its syntax. The language was defined with BNF (Bakus Normal Form or Bakus-Naur Form) [157]:

- Terminal symbols (keywords and operator symbols) are in a CONSTANT-WIDTH TYPEFACE.

- Choices are separated by vertical bars (|) and in greater-than and less-than symbols (< CHOICE>).
- Optional elements are in square brackets ([optional]).
- Sets of values are in curly braces ({SET}).
- A syntax description is introduced with ::=.

The language has three types of commands: **Team commands**, **Player commands** and **Game commands**. This way, a language is defined to be a set of commands that can be a TEAM_COMMAND, a GAME_COMMAND or a PLAYER_COMMAND. The TEAM_COMMAND is composed of the following ones: KICK_OFF, CORNER, THROW_IN, GOAL_KICK, FREE_KICK, PENALTY, GOAL or DROP_BALL. A GAME_COMMAND can be the START or STOP of the game, a command to end the game (END_GAME), cancel the just defined command (CANCEL) or resend the last command (RESEND). For the END_GAME command, it is necessary to define the game part, identified by PART_ID with one of four commands – 1ST, 2ND, EXTRA or PEN (penalties).

```
<LANGUAGE>::={<COMMAND>}

<COMMAND>::=<TEAM_COMMAND> | <GAME_COMMAND> | <PLAYER_COMMAND>

<TEAM_COMMAND>::=<KICK_OFF> | <CORNER> | <THROW_IN> | <GOAL_KICK> |
                 <FREE_KICK> |  <PENALTY> | <GOAL> | <DROP_BALL>

<GAME_COMMAND>::=<START> | <STOP> | <END_GAME> | <CANCEL> | <RESEND>

<PLAYER_COMMAND>::=<SUBSTITUTION> | <PLAYER_IN> | <PLAYER_OUT> |
                   <YELLOW_CARD> | <RED_CARD>
```

For the TEAM_COMMANDS there are several options: KICK_OFF, CORNER, THROW_IN, GOAL_KICK, FREE_KICK, PENALTY and GOAL that need a TEAM_ID (team identification) command, that can be one of two values - CYAN or MAGENTA, and finally the DROP_BALL command.

```
<KICK_OFF> ::= KICK_OFF <TEAM_ID>
<CORNER> ::= CORNER <TEAM_ID>
<THROW_IN> ::= THROW_IN <TEAM_ID>
<GOAL_KICK> ::= GOAL_KICK <TEAM_ID>
<FREE_KICK> ::= FREE_KICK <TEAM_ID>
<PENALTY> ::= PENALTY <TEAM_ID>
```

<GOAL> ::= GOAL <TEAM_ID>
<DROP_BALL> ::= DROP_BALL

For the PLAYER_COMMAND, first there is a SUBSTITUTION command with the identification of the player out (PLAYER_OUT) and the player in (PLAYER_IN) the game with the PLAYER_ID command. The PLAYER_ID can take one of seven values (PL1, PL2, PL3, PL4, PL5, PL6, PL7). For the remaining commands, PLAYER_IN, PLAYER_OUT, YELLOW_CARD or RED_CARD, it is necessary to define the TEAM_ID as explained above, and the PLAYER_ID.

<SUBSTITUTION> ::= SUBSTITUTION <PLAYER_IN> <PLAYER_OUT>
<PLAYER_IN> ::= PLAYER_IN <TEAM_ID> <PLAYER_ID>
<PLAYER_OUT> ::= PLAYER_OUT <TEAM_ID> <PLAYER_ID>
<YELLOW_CARD> ::= YELLOW_CARD <TEAM_ID> <PLAYER_ID>
<RED_CARD> ::= RED_CARD <TEAM_ID> <PLAYER_ID>
<START> ::= START
<STOP> ::= STOP
<END_GAME> ::= END_GAME <PART_ID>
<CANCEL> ::= CANCEL
<RESEND> ::= RESEND
<TEAM_ID > ::= CYAN | MAGENTA
<PLAYER_ID> ::= PL1 | PL2 | PL3 | PL4 | PL5 | PL6 | PL7
<PART_ID> ::= 1ST | 2ND | EXTRA | PEN

5.4.3 Prototype Implementation

The Human-Computer Interface for the prototype was implemented using the same previous tools. Also, for SVM training and classification the Dlib library was used, and for HMM training and classification the *openFrameworks* addon, *ofxSequence*, was used. This addon is a C++ porting of a MatLab code from Kevin Murphing (HMM MatLab Toolbox) [160].

The proposed system involves two modules, as shown in the diagram of Figure 46:

1. data acquisition, pre-processing and feature extraction
2. *gesture classification* with the models obtained in section 3.3 and 3.4

As explained in section 5.4.1, a referee command is composed by a sequence of dynamic gestures (Figure 47) and a set of static gestures (Figure 48). The static gestures are used to identify one of the following commands: team number, player number or game part.

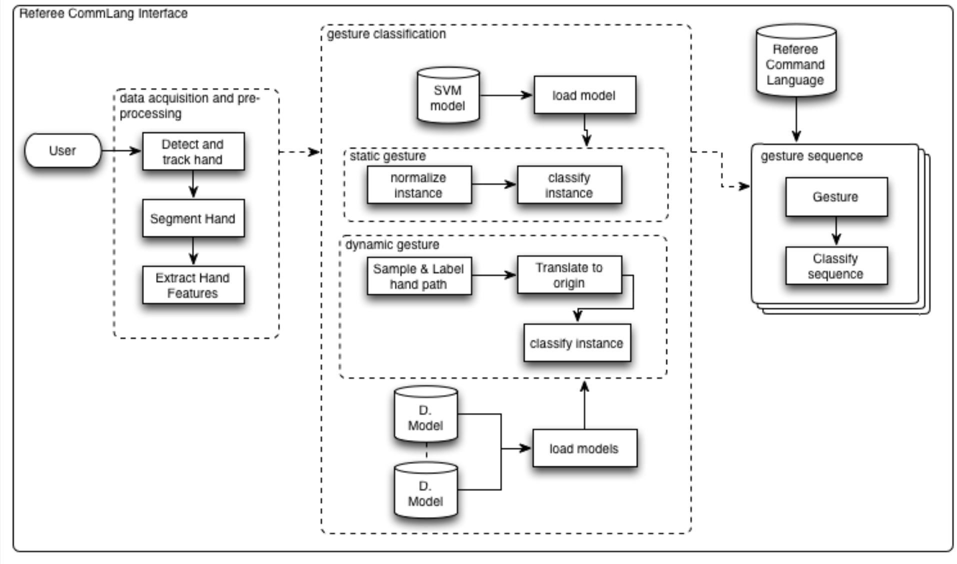


Figure 46. Referee CommLang Interface diagram with all the modules

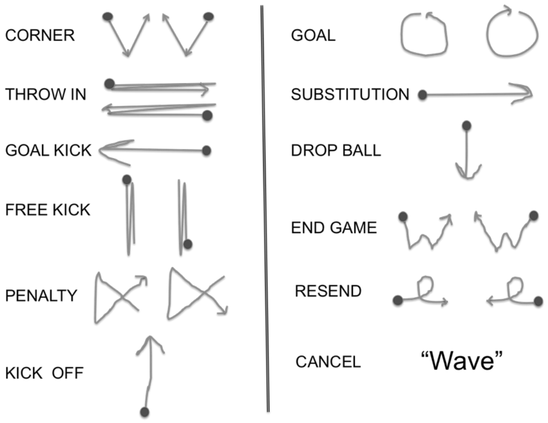


Figure 47. The set of dynamic gestures defined and used in the *Referee CommLang*.

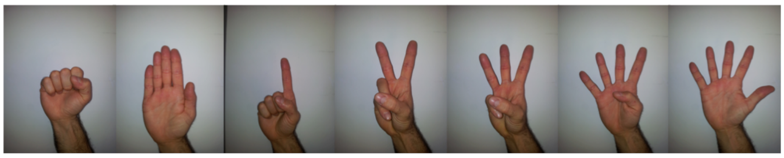


Figure 48. The set of hand postures trained and used in the *Referee CommLang.*

The following sequence of images shows the Referee Command Language user interface with the "GOAL, TEAM1, PLAYER2" sequence of commands being recognized.

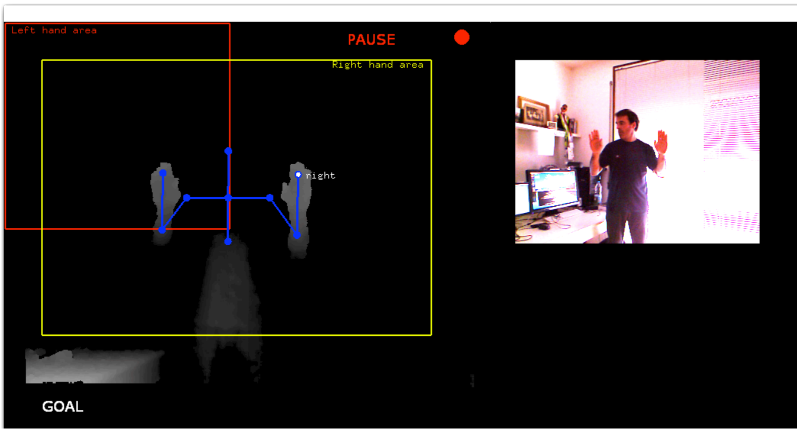


Figure 49. The "GOAL" gesture recognized.

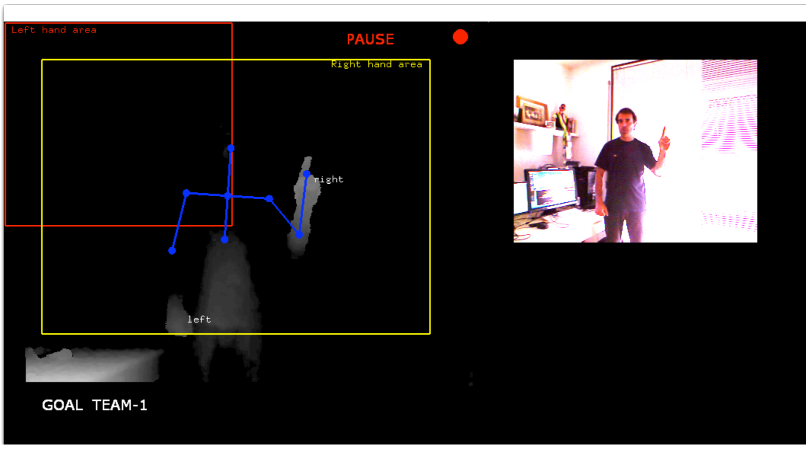


Figure 50. The "GOAL, TEAM1" sequence recognized.

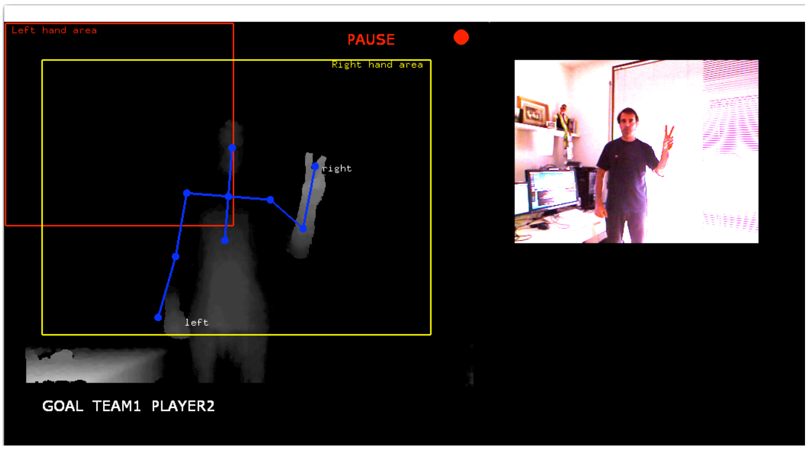


Figure 51. The "GOAL, TEAM1, PLAYER2" sequence recognized.

In the following sequence of images, another sequence of commands is being recognized: the “SUBSTUTUTION, TEAM1, PLAYER-IN-1, PLAYER-OUT-3”.

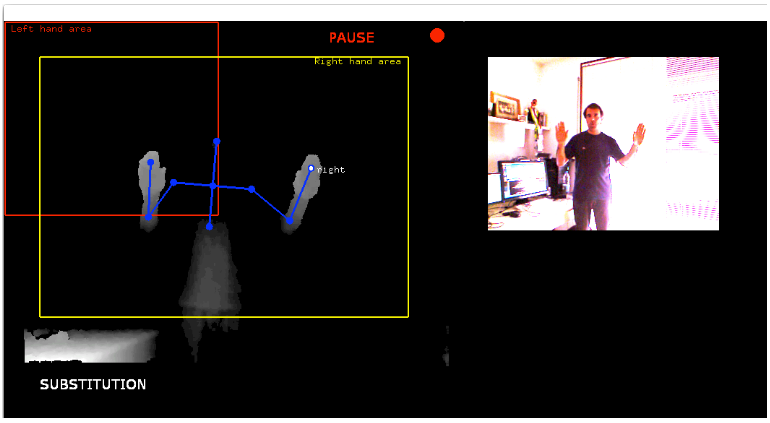


Figure 52. The "SUBSTITUTION" gesture recognized.

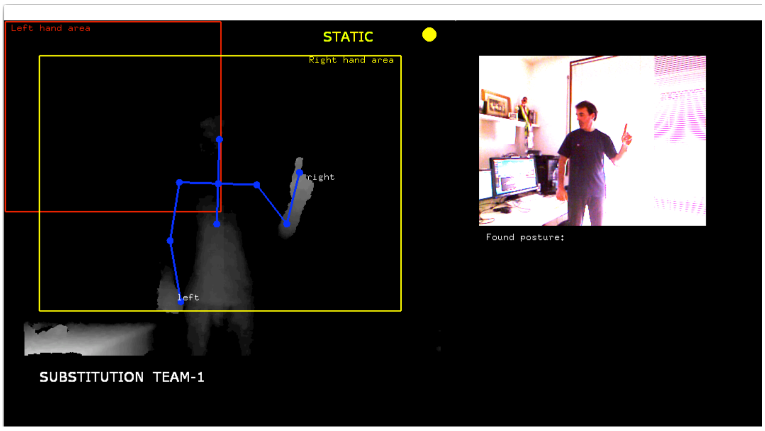


Figure 53. The "SUBSTITUTION TEAM-1" sequence recognized.

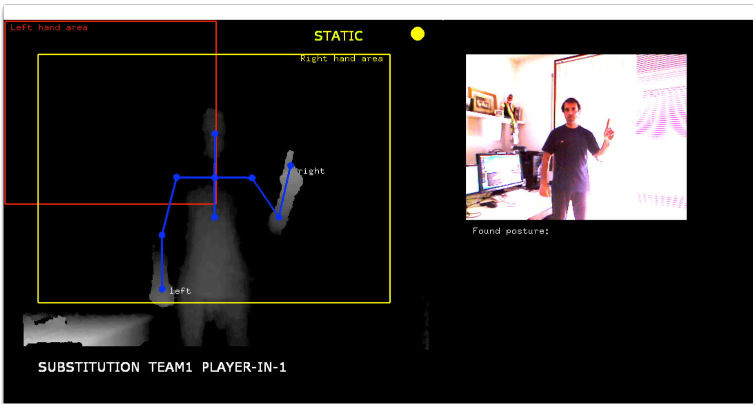
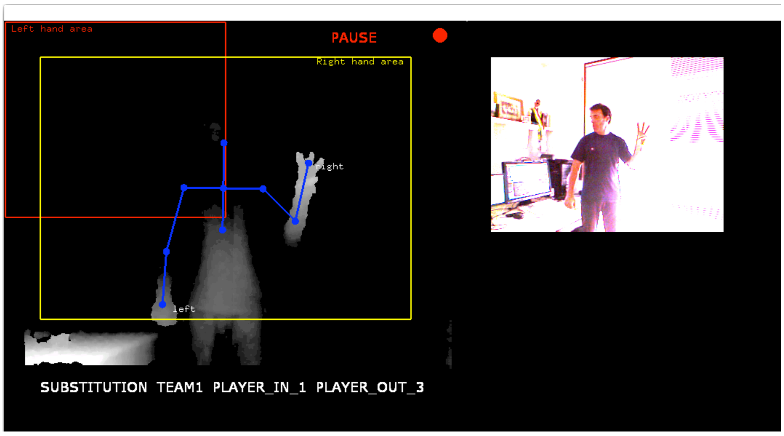


Figure 54. The "SUBSTITUTION TEAM-1 PLAYER-IN-1" sequence recognized



SUBSTITUTION TEAM1 PLAYER_IN_1 PLAYER_OUT_3

Figure 55. The "SUBSTITUTION TEAM-1 PLAYER-IN-1 PLAYER-OUT-3" sequence recognized

5.5 Sign Language Recognition Prototype

5.5.1 Introduction

The *Sign Language Recognition Prototype* is a real-time vision-based system whose purpose is to recognize the Portuguese Sign Language given in the alphabet of Figure 57. The purpose of the application was to test two types of hand features and verify, with the models learned, their performance in terms of real-time classification.

For that, the user must be positioned in front of the camera, doing the sign language gestures, that will be interpreted by the system and their classification will be displayed on the right side of the interface.

The diagram from Figure 56 shows the modules that compose the proposed system architecture. It is manly composed of two modules:

1. data acquisition, pre-processing and feature extraction
2. sign language gesture classification

At the moment the system is trained to recognize only the vowels, but it is easily extended to recognize the rest of the alphabet.

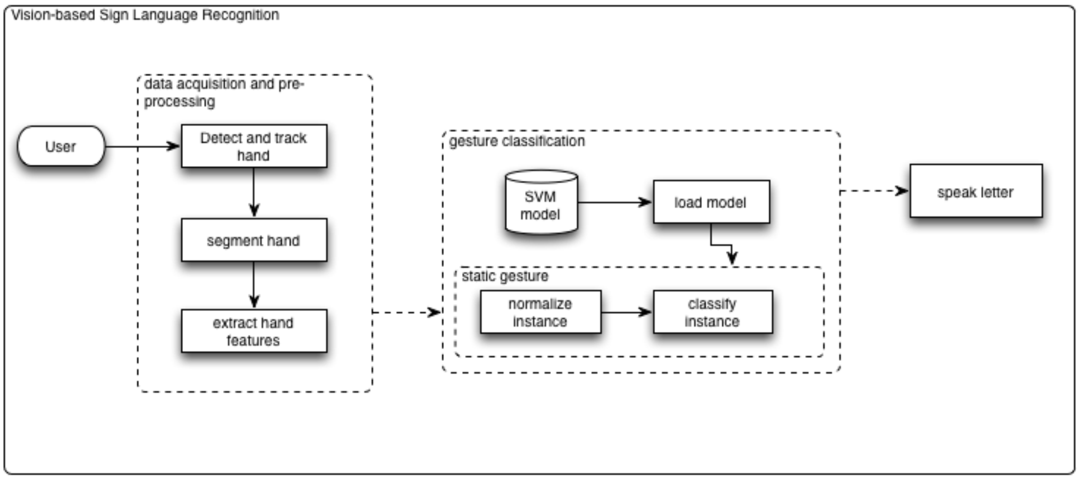


Figure 56. Sign Language Recognition Prototype diagram

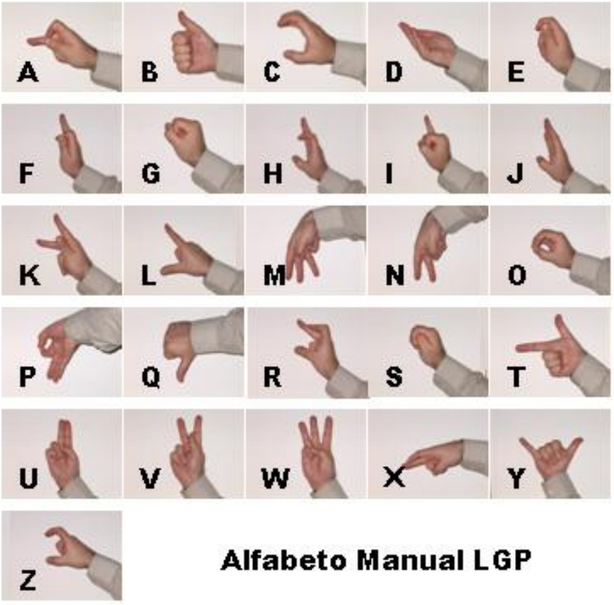


Figure 57. Portuguese Language manual alphabet

5.5.2 Prototype Implementation

The human-computer interface for the prototype was developed using the C++ language, and the openFrameworks toolkit with the following two addons: the OpenCV and the OpenNI addons, *ofxOpenCv* and *ofxOpenNI* respectively. Also, for SVM training and classification the application uses the Dlib library.

In the following sequence of images it is possible to see the Sign Language Prototype with all the vowels correctly classified.

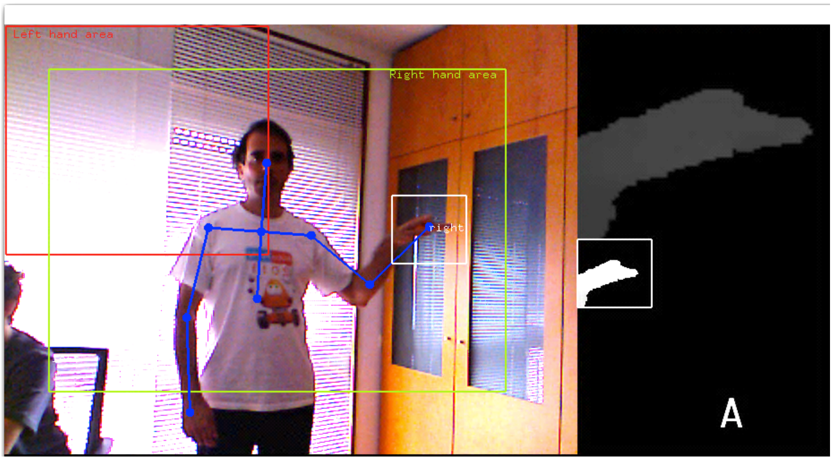


Figure 58. The vowel A correctly classified

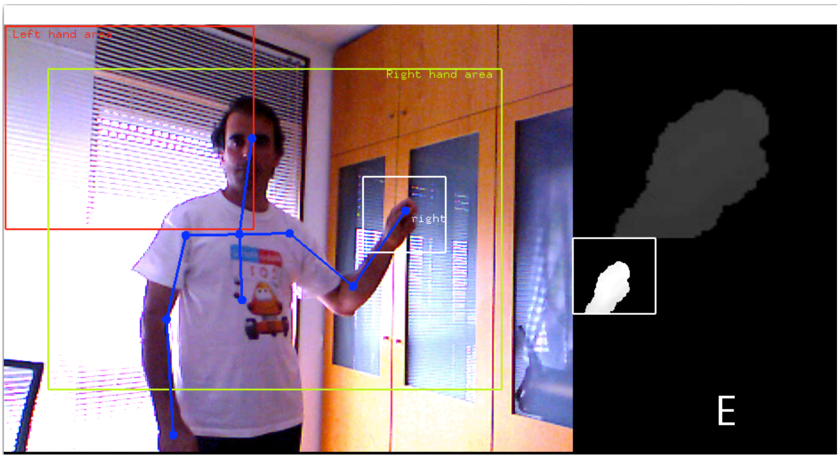


Figure 59. The vowel E correctly classified

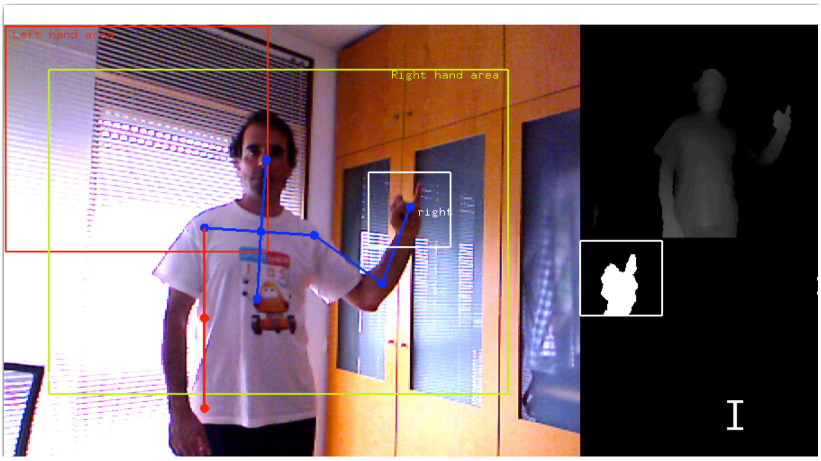


Figure 60. The vowel I correctly classified

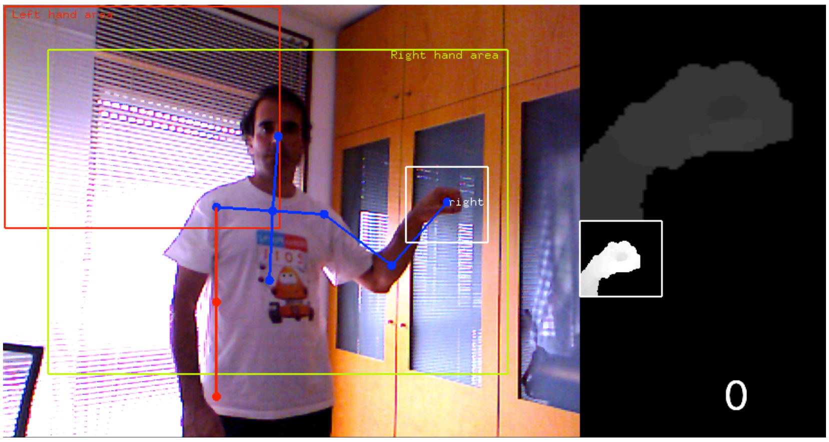


Figure 61. The vowel O correctly classified

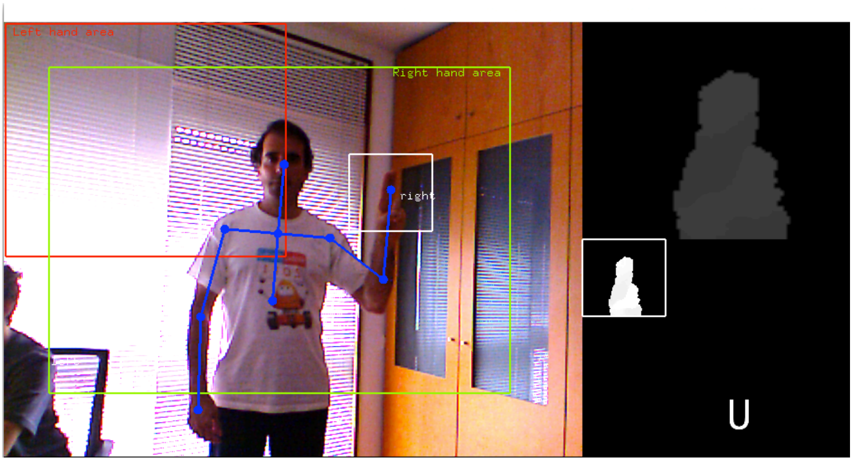


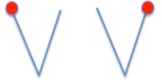
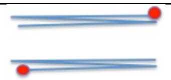
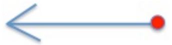
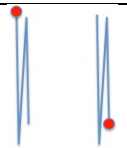
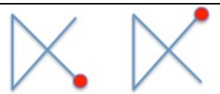
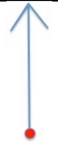
Figure 62. The vowel U correctly classified

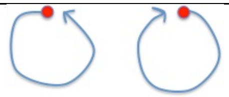
5.6 Summary

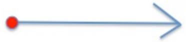
This chapter presented a set of applications that were implemented during the study namely: *the vision-based remote hand robot control system*, *the vision-based hand robotic wheel-chair control system*, *the Referee CommLang Prototype* system and *the Sign Language Recognition Prototype*. The first one is a system that enables a user to remotely control a robot / system with a number of simple hand commands made in front of a camera. The system uses the segmented hand information to derive heading and also velocity values that are transmitted to the robot. The second one is a system that enables a user to drive a robotic base wheelchair with a minimum number of finger commands made in front of a camera. Once again, the segmented hand is used to extract the finger information and build the proper commands that are transmitted to the robot. A new formal language definition for the finger commands was presented. The third is a real-time vision-based system that is able to interpret a set of commands defined for the MSL referee and transmit them to the RefereeBox. For that, a new formal language definition was also presented, capable of representing all the necessary defined RefereeBox commands.

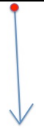
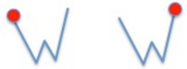
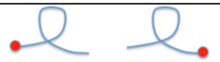
The last system is a prototype for Portuguese Sign Language recognition. Although it is only trained to identify the vowels at the moment, the system is easily extended to recognize the rest of the alphabet. The reason for working with vowels only is explained by the fact that the main goal was to validate the implemented solutions in another type of human-computer interaction application, and so, only a few gestures were trained. It was however possible to verify the validity of the proposed methods in another type of real-time human-computer interface, where the core of the vision-based interaction is the same for all applications.

Table 4. Command set definition with associated odes and text description

Nº	Command	1st Gesture	2nd Gesture	3rd Gesture	4th Gesture	Code (TEXT)
1	CORNER		TEAM			**11** – CORNER, TEAM 1 **12** – CORNER, TEAM 2
2	THROW_IN		TEAM			**21** – THROW_IN, TEAM 1 **22** – THROW_IN, TEAM 2
3	GOAL_KICK		TEAM			**31** – GOAL_KICK, TEAM 1 **32** – GOAL_KICK, TEAM 2
4	FREE_KICK		TEAM			**41** – FREE_KICK, TEAM 1 **42** – FREE_KICK, TEAM 2
5	PENALTY		TEAM			**51** – PENALTY, TEAM 1 **52** – PENALTY, TEAM 2
6	KICK_OFF		TEAM			**61** – KICK_OFF, TEAM 1 **62** – KICK_OFF, TEAM 2

7	GOAL		TEAM	PLAYER		**71(1-7)** – GOAL, TEAM1, PLAYER(1-7)
						72(1-7) – GOAL, TEAM2, PLAYER(1-7)

8	SUBSTITUTION		TEAM	PLAYER_IN	PLAYER_OUT	**81(1-7)(1-7)** – SUBSTITUTION, TEAM 1, PLAYER_IN(1-7), PLAYER_OUT(1-7)
						82(1-7)(1-7) – SUBSTITUTION, TEAM 2, PLAYER_IN(1-7), PLAYER_OUT(1-7)

9	DROP_BALL					**9** - DROP_BALL
10	END_GAME					**101** – END_GAME, PART 1 **102** – END_GAME, PART 2
11	RESEND					**11** – CANCEL
12	RESEND	"Wave"				**12** - RESEND

6 Experiments and Results

6.1 Introduction

In chapter 3, the *Gesture Learning Module Architecture (GeLMA*) and the connection between its various modules was described. Some of the requisites that a system based on vision for human-robot interaction must satisfy, in order to be successfully implemented, were listed. It was also mentioned that, to implement such a system, it is necessary to be able to learn models that can be used in real-time gesture classification situations.

The aim of the proposed system is to enable the recognition of static or dynamic gestures or a combination of both. Thus, in order to select a set of hand features that could meet the requirements of robustness, computationally efficiency, error tolerance and scalability, a set of experiments were performed with hand features collected from a set of users who executed the pre-defined gestures in front of a Kinect camera. The extracted features were used alone or combined, in order to find which of them behaved better within a pre-established set of parameters. Those experiments were performed with the help of the RapidMiner Community Edition [161], in order to select machine learning algorithms that would achieve the best classification results for the given datasets. RapidMiner is an open-source data mining solution that enables to explore data and at the same time simplify the construction of analysis processes and the evaluation of different approaches. It has more than 400 data-mining operators that can be used and almost arbitrarily combined. This way, RapidMiner can be seen as integrated development environment (IDE) for machine learning and data mining, and a valuable tool in this field.

Experiments were also performed with dynamic gestures features in order to learn HMM parameters, and build classifiers able to do real-time dynamic gesture recognition. For the HMM model learning and implementation the Dlib library [145], a general purpose cross-platform C++ library, was used. This is a library for developing portable applications dealing within a number of areas, including machine learning.

The following sections describe in detail each of the experiments and present the results in order to prove that the objectives of this work (section 1.2) were achieved in order to validate the approach and the adopted methodology.

The presentation will be carried out in accordance with the various aspects and developments of the implemented prototypes. With this approach, it is intended to show an evolution of the choices made and the results achieved during the work progress.

6.2 Comparative study of machine learning algorithms for hand posture classification

6.2.1 Experimental Setup

This experiment intended to carry a comparative study of the following four machine learning algorithms, *k-Nearest Neighbour (k-NN), Naïve Bayes (NB), Artificial Neural Network (ANN) and Support Vector Machines (SVM)*, applied to two datasets composed of different sets of hand features. The main goal of the experiment was to understand if machine-learning algorithms could improve and could be used efficiently in real-time human-computer interaction systems applied to hand posture classification.

For that, two datasets were used composed of different features and analysed the efficiency of each ML algorithm in terms of its recognition rate accuracy. The first dataset was composed by the following features: *hand angle*, *mean and variance* of the segmented grey image hand, *36 values from the orientation histogram*, *and 100 values from the hand radial signature*. The second dataset was composed by the following features: *hand angle*, *mean and variance of the segmented hand grey image*, *area and perimeter of the binary hand blob* and *the number of convexity defects*. A convexity defect is "*another useful way of comprehending the shape of an object by its convex hull and then compute its convexity defects*" [28]. Figure 63 illustrates the concept of convexity defects using a hand image, with the corresponding convex hull represented as the dark line around the hand. The regions (A-H) are the convexity defects in the hand contour.

Some samples from the two datasets, before normalization with the *z-normalization* method (section 3.3.1), are represented on the next two tables. The total number of records used in the first dataset was 1295 with ten representative classes, and in the second dataset were 945, also with ten representative classes. The features were extracted with the help of four users and in a laboratory controlled environment.

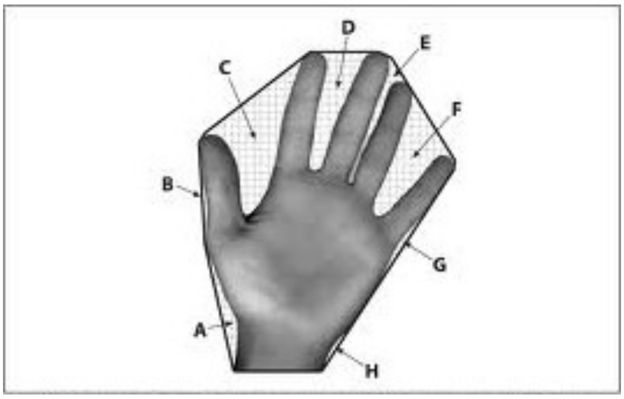


Figure 63. Hand *convexity defects* [28].

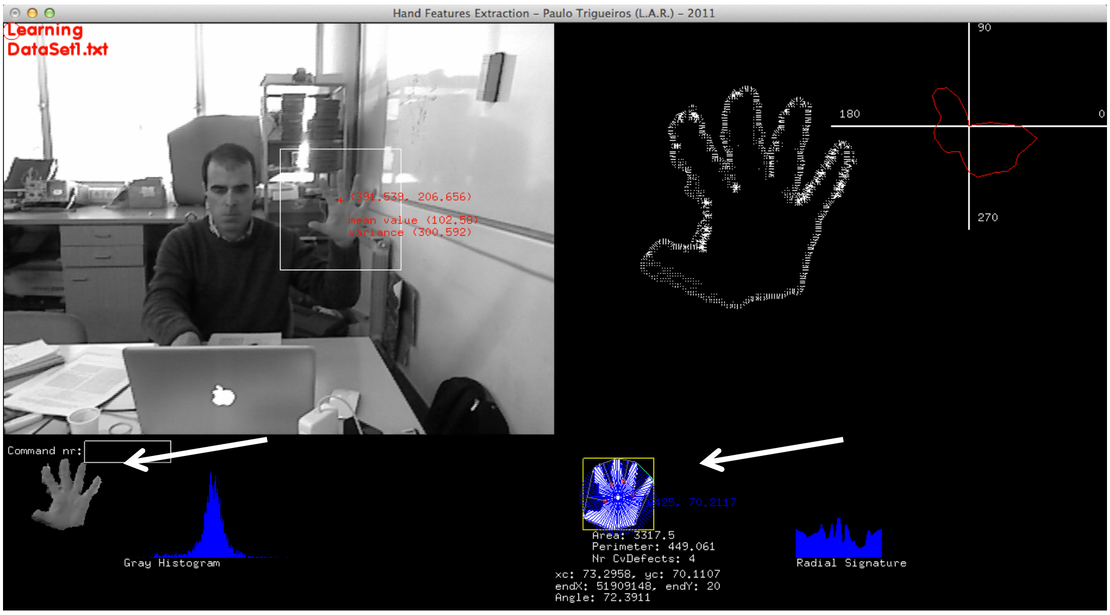
Table 5. Sample values for the 1st dataset prior to vector normalization.

Class	Angle	Mean	Var.	Hog-1	Hog-2	…	Rs-99	Rs-100
1	0.123	0.038	1	0.109	0.140	…	0.022	0.028
1	0.131	0.037	1	0.107	0.142	…	0.023	0.025
2	0.173	0.076	1	0.284	0.398	…	0.110	0.110
2	0.174	0.076	1	0.255	0.374	…	0.106	0.108
3	0.147	0.063	1	0.124	0.170	…	0.032	0.032
3	0.144	0.065	1	0.106	0.144	…	0.018	0.018
4	0.119	0.053	1	0.125	0.183	…	0.054	0.055
4	0.126	0.059	1	0.120	0.185	…	0.044	0.043
...	...	...	...	...	...	…	...	...

Table 6. Sample values for the 2nd dataset prior to vector normalization.

Class	Angle	Mean	Variance	Area	Perimeter	Convexity Defects
1	0.034	0.009	0.265	1	0.077	0
1	0.035	0.009	0.263	1	0.082	0.0003
2	0.027	0.014	0.152	1	0.075	0
2	0.028	0.014	0.164	1	0.075	0.0003
3	0.047	0.047	1	0.878	0.104	0.0004
3	0.043	0.045	1	0.841	0.099	0.0004
4	0.033	0.045	1	0.816	0.102	0.0003
4	0.044	0.065	0.944	1	0.133	0.0004
...	...	...	...	...	...	...

Since the first step in the experimental setup was the creation of the datasets with the corresponding hand features, a C++ application was created (Figure 64), able to interface with a Kinect camera and extract all the necessary information. For hand feature extraction, the corresponding grey image hand (bottom left) and the binary hand image (bottom centre) were used.



```
(391.539, 206.656)
mean value (102.58)
variance (300.592)
```

```
425, 70.2117
Area: 3317.5
Perimeter: 449.061
Nr CvDefects: 4
xc: 73.2958, yc: 70.1107
endX: 51909148, endY: 20
Angle: 72.3911
```

Figure 64. Application user interface used for hand feature extraction.

In the image, it is possible to observe the segmented grey hand with the corresponding histogram below the main camera image. To the right it can be seen the hand binary blob with radials, used to compute the radial signature histogram, drawn on top of it. Below the hand blob it is the respective hand information: *area*, *perimeter*, *number of convexity defects* and *angle*. To the right of the hand blob the radial signature histogram is displayed. On the top right side, on the right of the camera image, the hand gradients and the respective HoG (Histogram of Gradients)can be seen. Displayed on the camera image, on the top left of the application interface, there is the information concerning system status. In the present case the system is in a learning state, and recording values for the first dataset.

After dataset creation, the obtained files are converted to Excel files in order to be imported to RapidMiner for algorithm performance testing, parameter optimization and learner selection. As explained before, the four chosen algorithms (k-NN, Naïve Bayes, ANN, SVM) were applied to the two datasets, and the experiments were performed under the assumption of the *k-fold cross validation method*. The *k-fold cross validation* is used to determine how accurately a learning algorithm will be able to predict data not trained with [68]. In the k-fold cross-validation, the dataset X is divided randomly into k equal sized parts, X_i : $i =1,..., k$. The learning algorithm is then trained k times, using k-1 parts as the training set and the one that stays out as the validation set. A value of k=10 is normally used, giving a good rule of approximation, although the best value depends on the used algorithm and the dataset [66, 77]. As explained by Ian H. Witten et al. [77], "*extensive tests on numerous different datasets, with different learning techniques, have shown that 10 is about the right number of folds to get the best estimate of errors*".

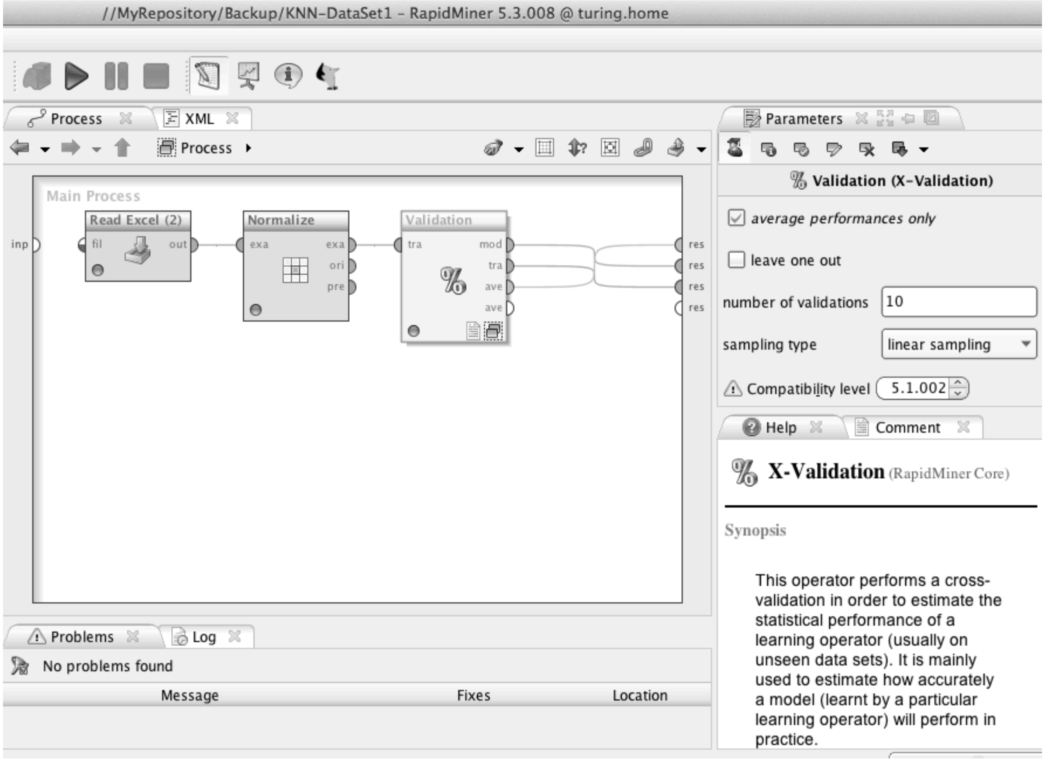
Prior to learning and model application, the data was normalized as explained before. Finally, a performance test was carried out, based on the number of counts of test records correctly and incorrectly predicted by the model.

Since classifier settings and parameters used are important aspects to take into account, for all the algorithms a parameter optimization analysis was carried out.

For the simplest of the algorithms, the k-NN, a value of k=1 (number of neighbours used) for the two datasets was obtained with an Euclidean distance metric.

The following images show RapidMiner process setup for the k-NN dataset 1 analysis with the corresponding setups. The first image has represented the file import phase, data normalization and cross-validation configuration, with the number of validations set to 10, as explained before. For the sampling type there are three possible options: *linear*, *shuffled* and *stratified*. Linear sampling simply divides the dataset into partitions without changing the order, i.e. subsets with consecutive samples are created. The shuffled sampling builds random subsets with the dataset, i.e., examples are chosen randomly for making subsets. Stratified sampling builds random subsets and ensures that the class distribution in the subsets is the same as in the whole dataset, i.e. each subset contains roughly the same proportions of the number of classes. This parameter has not been changed during the tests, where the default value was accepted.

The second image is the actual k-NN learning process configuration, model application and performance testing. For the k-NN algorithm, as it can be seen on the right side, the k is set to one and an Euclidean distance measure is applied.



☑ average performances only

☐ leave one out

number of validations: 10

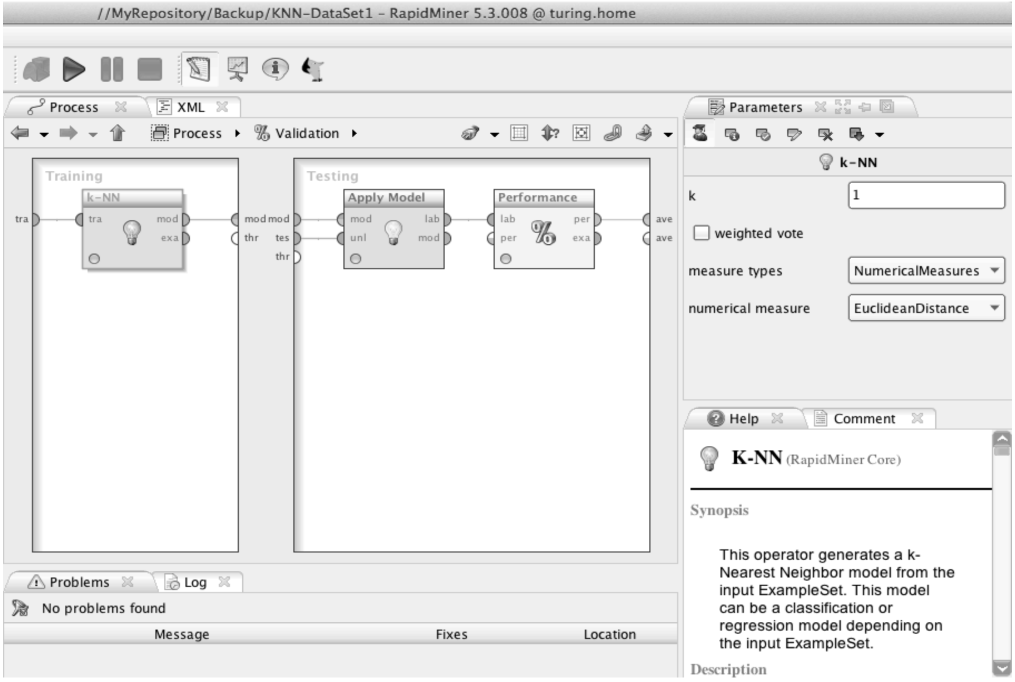
sampling type: linear sampling

Compatibility level: 5.1.002

No problems found

This operator performs a cross-validation in order to estimate the statistical performance of a learning operator (usually on unseen data sets). It is mainly used to estimate how accurately a model (learnt by a particular learning operator) will perform in practice.

Figure 65. First RapidMiner setup screen for the k-NN dataset 1 analysis.



//MyRepository/Backup/KNN-DataSet1 – RapidMiner 5.3.008 @ turing.home

k: 1

☐ weighted vote

measure types: NumericalMeasures

numerical measure: EuclideanDistance

No problems found

K-NN (RapidMiner Core)

Synopsis

This operator generates a k-Nearest Neighbor model from the input ExampleSet. This model can be a classification or regression model depending on the input ExampleSet.

Description

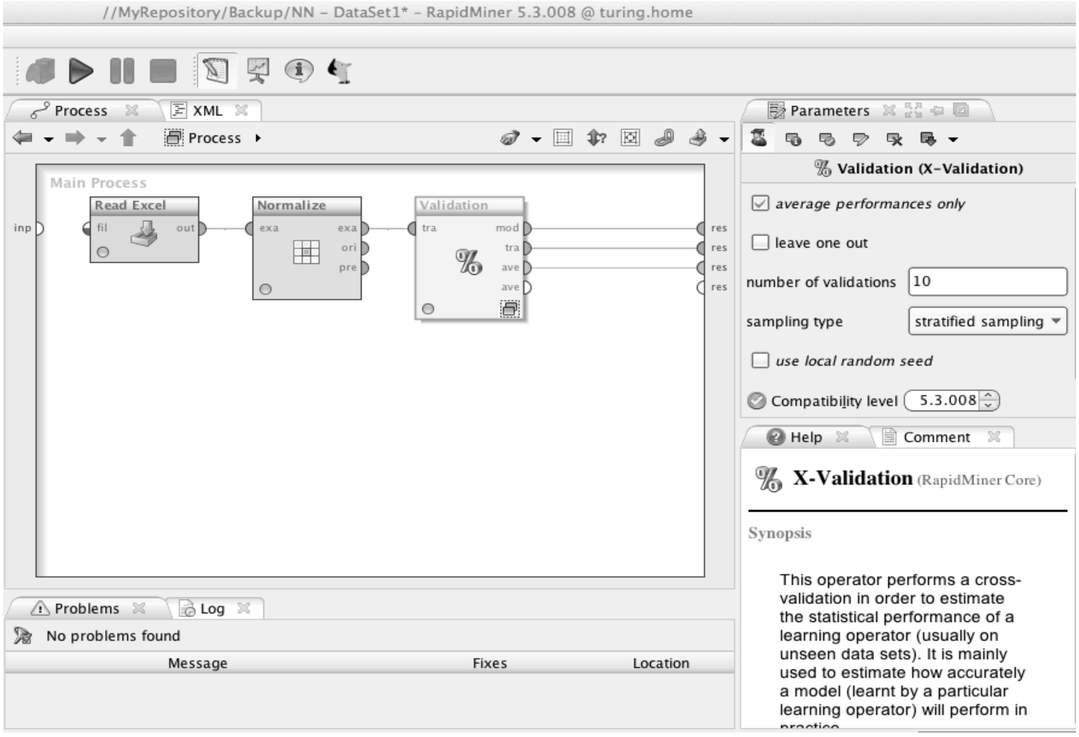
Figure 66. Second RapidMiner setup screen for the k-NN dataset 1 analysis (training and testing).

For the artificial neural network, the number of training cycles was defined as constant and equal to 500. The learning rate, and the momentum were obtained via parameter optimization. The *learning rate* is user-designated and used to determine how much the weights can be modified based on the change direction and change rate. The higher the learning rate, the faster the network is trained. *Momentum* basically allows a change to the ANN weights, during learning, to persist for a number of adjustment cycles. It is used to prevent the system from converging to a local minimum or saddle point [76, 80]. The following table shows the obtained values for the two datasets, after parameter optimization.

Table 7. Artificial Neural Network (ANN) parameter setup used.

Parameters	**Values**	
	Dataset 1	Dataset 2
Learning rate	0.1	0.33
Momentum rate	0.1	0.18
Training cycles	500	500
Hidden layers in the network	1	1

In the following images, the RapidMiner processes setup configurations for the two datasets are represented. The first image, common to all configurations, has the file import, data normalization and cross validation setup. In the second image, the actual ANN learning process configuration, model application and performance testing are configured. For the ANN algorithm, as can be seen on the parameter definition of Figure 68, the learning rate and the momentum are both set to 0.1, value obtained during the parameter optimization phase with the best accuracy results.



Validation (X-Validation)

☑ average performances only

☐ leave one out

number of validations: 10

sampling type: stratified sampling

☐ use local random seed

Compatibility level: 5.3.008

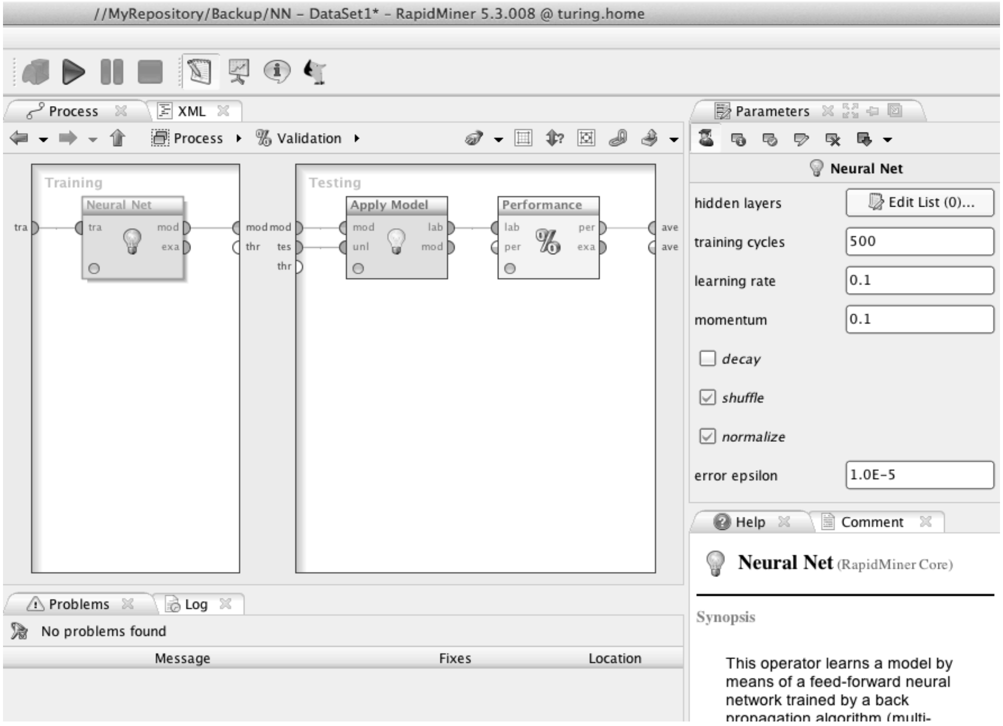
No problems found

X-Validation (RapidMiner Core)

Synopsis

This operator performs a cross-validation in order to estimate the statistical performance of a learning operator (usually on unseen data sets). It is mainly used to estimate how accurately a model (learnt by a particular learning operator) will perform in

Figure 67. First RapidMiner setup screen for the ANN dataset 1 analysis.



//MyRepository/Backup/NN – DataSet1* – RapidMiner 5.3.008 @ turing.home

No problems found

Neural Net

hidden layers: Edit List (0)...

training cycles: 500

learning rate: 0.1

momentum: 0.1

☐ decay

☑ shuffle

☑ normalize

error epsilon: 1.0E-5

Neural Net (RapidMiner Core)

Synopsis

This operator learns a model by means of a feed-forward neural network trained by a back propagation algorithm (multi-

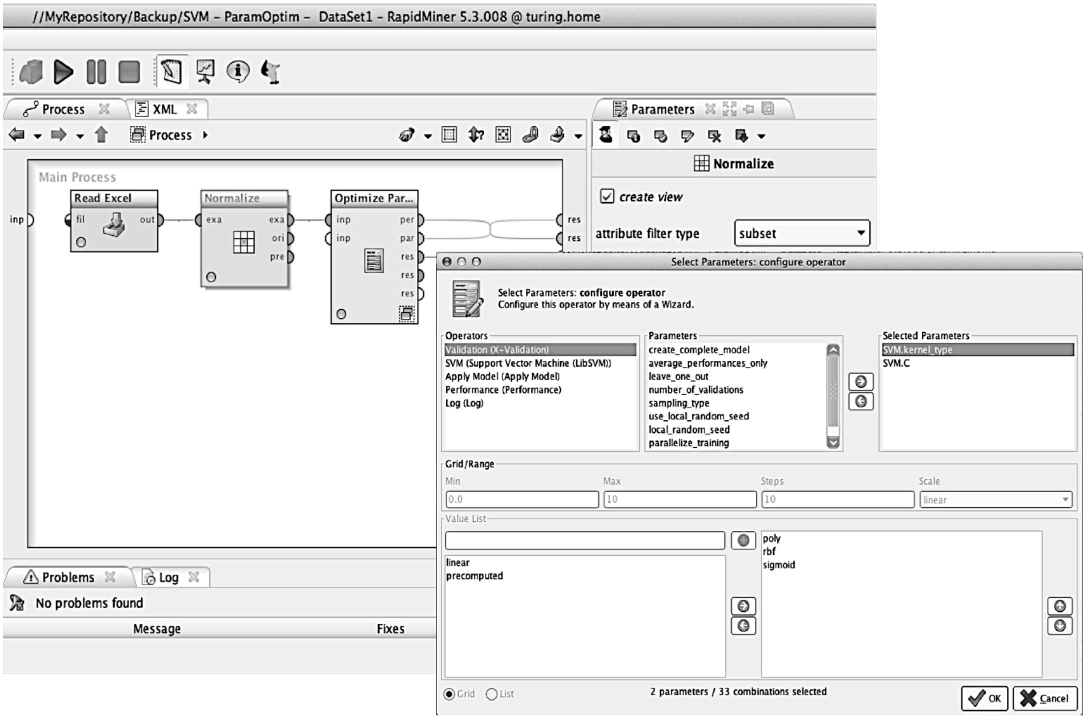
Figure 68. RapidMiner ANN DataSet1 analysis (second screen).

For the SVM tests, the RapidMiner libSVM [162] algorithm implementation was used, which supports multiclass learning with the "*one-against-one*" approach [163, 164]. The type of SVM used was the C-SCV, and the kernels obtained after parameter optimization were the *sigmoid* for dataset 2 with a C value equal to 0 and the *rbf* (*radial basis function*) kernel with a C value equal to 10 for dataset 1 as shown on the table below.

Table 8. Parameters obtained after SVM optimization process

Parameters	Dataset 1	Dataset 2
Kernel type	rbf	Sigmoid
Cost parameter - C	10	0

The RapidMiner process setup screens are shown in the following two images. Figure 69 shows the parameter optimization configuration window, where the type of kernels for optimization are defined, that in the setup were the poly, rbf and sigmoid. For the C parameter, values in the range 1 to 10 with a linear scale were defined. Figure 70 shows the actual SVM learning process configuration and the testing phase where the obtained model is tested and the performance value returned. For the SVM algorithm, as can be seen on the parameter definition on Figure 70, the kernel type was set to *rbf* and the C value was set to 10, as obtained during the optimization phase.



//MyRepository/Backup/SVM – ParamOptim – DataSet1 – RapidMiner 5.3.008 @ turing.home

Normalize

☑ create view

attribute filter type: subset

Select Parameters: configure operator

Select Parameters: configure operator
Configure this operator by means of a Wizard.

Operators	Parameters	Selected Parameters
Validation (X-Validation)	create_complete_model	SVM.kernel_type
SVM (Support Vector Machine (LibSVM))	average_performances_only	SVM.C
Apply Model (Apply Model)	leave_one_out	
Performance (Performance)	number_of_validations	
Log (Log)	sampling_type	
	use_local_random_seed	
	local_random_seed	
	parallelize_training	

Grid/Range

Min	Max	Steps	Scale
0.0	10	10	linear

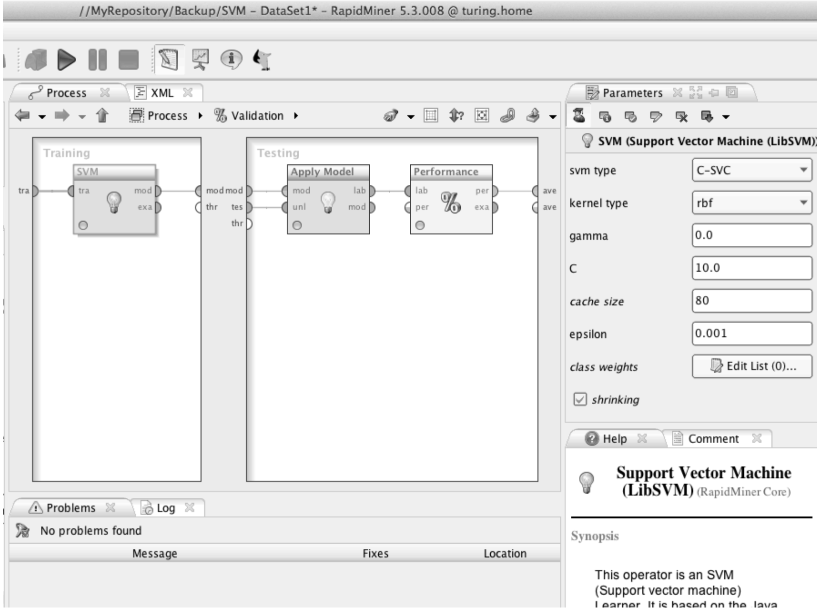
Value List: linear, precomputed | poly, rbf, sigmoid

● Grid ○ List

2 parameters / 33 combinations selected

No problems found

Figure 69. RapidMiner parameter optimization setup for the SVM dataset 1 analysis.



//MyRepository/Backup/SVM – DataSet1* – RapidMiner 5.3.008 @ turing.home

SVM (Support Vector Machine (LibSVM))

svm type: C-SVC
kernel type: rbf
gamma: 0.0
C: 10.0
cache size: 80
epsilon: 0.001
class weights: Edit List (0)...
☑ shrinking

Support Vector Machine (LibSVM) (RapidMiner Core)

Synopsis

This operator is an SVM (Support vector machine) Learner. It is based on the Java

No problems found

Figure 70. Second RapidMiner screen setup for the SVM dataset 1 analysis.

6.2.2 Results

The results obtained with the two datasets are represented in Table 9. Here one can observe the obtained accuracy in each algorithm for both datasets and the respective time spent on the tests.

Table 9. Accuracy obtained on both datasets with each algorithm, and time spent in the tests.

	Classifier	**k-NN**	**Naïve Bayes**	**ANN**	**SVM**
Dataset 1	Accuracy **(%)**	**95,45**	25,87	**96,99**	**91,66**
	Time	8s	1s	46m33s	3m10s
Dataset 2	Accuracy **(%)**	88,52	66,50	85,18	80,02
	Time	1s	1s	32s	1m08s

To analyse how classification errors are distributed among classes a confusion matrix for each trained algorithm was calculated, and the obtained results are shown on Table 10 through Table 17.

In the field of machine learning, a *confusion matrix* is a table layout that allows visualization of an algorithm performance, typically a supervised learning one. In RapidMiner, each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class. The name, confusion matrix, stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabelling one as another) [66, 77, 165].

The confusion matrix accuracy is calculated according to the following formula:

$$accuracy = \sum a_{ii} / \sum a_{ij}\,,\ i = 1, \dots, n; j = 1, \dots, n \tag{30}$$

Table 10. k-NN confusion matrix for dataset 1

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**91**	1	0	0	1	0	0	0	0	1
	2	0	**131**	0	0	0	1	1	0	0	5
	3	1	0	**122**	1	0	0	0	2	0	1
	4	0	0	0	**102**	3	1	0	0	0	0
	5	0	0	1	2	**123**	0	0	0	0	0
	6	0	0	0	0	0	**122**	7	0	0	0
	7	0	0	0	0	0	5	**120**	0	0	0
	8	0	0	1	0	0	0	0	**146**	0	0
	9	1	0	0	0	0	0	0	0	**155**	0
	10	0	2	0	1	0	0	0	0	0	**144**

Table 11. k-NN confusion matrix for dataset 2

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**59**	1	2	0	0	0	0	0	4	1
	2	0	**74**	0	0	0	0	1	0	0	1
	3	2	0	**49**	1	2	0	0	3	1	3
	4	0	0	1	**38**	10	0	0	1	0	2
	5	0	0	0	6	**52**	6	2	0	0	2
	6	0	0	0	0	5	**68**	1	0	1	1
	7	0	0	0	0	0	1	**86**	0	0	0
	8	7	0	5	1	1	0	0	**92**	0	0
	9	0	0	0	0	0	1	0	0	**82**	0
	10	2	1	5	0	2	1	0	2	1	**87**

Table 12. Naïve Bayes confusion matrix for dataset 1

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**11**	4	6	0	0	1	7	7	0	1
	2	14	**49**	0	24	25	24	20	0	12	24
	3	25	4	**24**	27	66	55	54	7	65	21
	4	0	0	0	**0**	0	0	0	0	0	0
	5	0	10	0	0	**1**	14	9	0	0	2
	6	0	3	0	0	0	**0**	0	0	0	0
	7	6	6	18	4	4	6	**10**	8	16	0
	8	33	0	73	47	24	15	5	**123**	43	5
	9	3	0	3	0	0	0	1	3	**19**	0
	10	1	58	0	4	7	14	22	0	0	**98**

Table 13. Naïve Bayes confusion matrix for dataset 2

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**62**	0	0	0	0	0	0	3	11	1
	2	2	**74**	0	0	0	0	0	0	0	2
	3	2	0	**54**	1	0	0	0	7	2	5
	4	0	0	2	**34**	5	0	0	0	0	1
	5	0	0	1	8	**64**	8	2	0	2	2
	6	0	0	0	1	1	**67**	3	0	1	4
	7	0	0	0	0	1	2	**84**	0	0	1
	8	4	1	3	2	0	0	0	**79**	0	9
	9	0	0	0	0	0	0	0	0	**72**	1
	10	0	1	2	0	1	0	0	9	1	**71**

Table 14. ANN confusion matrix for dataset 1

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**91**	0	4	0	1	0	0	0	2	0
	2	0	**129**	0	0	1	1	2	0	0	1
	3	0	1	**117**	0	0	0	1	0	0	2
	4	0	0	3	**103**	1	0	0	0	0	0
	5	1	2	0	2	122	2	2	0	0	1
	6	0	0	0	0	1	125	0	0	0	0
	7	0	0	0	1	0	1	122	0	1	0
	8	1	0	0	0	0	0	0	148	0	0
	9	0	0	0	0	0	0	1	0	152	0
	10	0	2	0	0	1	0	0	0	0	147

Table 15. ANN confusion matrix for dataset 2

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**62**	0	0	0	0	0	0	3	11	1
	2	2	**74**	0	0	0	0	0	0	0	2
	3	2	0	**54**	1	0	0	0	7	2	5
	4	0	0	2	**34**	5	0	0	0	0	1
	5	0	0	1	8	**64**	8	2	0	2	2
	6	0	0	0	1	1	**67**	3	0	1	4
	7	0	0	0	0	1	2	**84**	0	0	1
	8	4	1	3	2	0	0	0	**79**	0	9
	9	0	0	0	0	0	0	0	0	**72**	1
	10	0	1	2	0	1	0	0	9	1	**71**

Table 16. SVM confusion matrix for dataset 1

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**85**	0	0	0	0	4	0	1	0	1
	2	0	**116**	0	4	0	1	1	0	0	3
	3	0	0	**115**	7	1	1	0	0	0	1
	4	0	0	3	**86**	11	3	1	0	0	1
	5	3	0	0	9	**112**	10	2	0	0	1
	6	0	2	0	0	0	**109**	4	0	0	0
	7	0	0	0	0	0	1	**119**	0	0	0
	8	4	0	6	0	0	0	0	**146**	0	0
	9	1	0	0	0	0	0	0	1	**155**	0
	10	0	16	0	0	3	0	1	0	0	**144**

Table 17. SVM confusion matrix for dataset 2

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**59**	0	0	0	0	0	0	0	10	0
	2	0	**65**	0	0	0	0	0	0	0	12
	3	3	0	**37**	0	1	1	0	6	7	2
	4	0	0	2	**33**	2	0	0	0	0	1
	5	0	0	0	10	**67**	9	2	0	0	5
	6	0	0	0	0	0	**64**	2	0	0	1
	7	0	0	0	0	0	3	**85**	0	0	0
	8	3	0	13	0	0	0	0	**74**	0	11
	9	0	0	0	0	1	0	0	1	**72**	0
	10	5	11	10	3	1	0	0	17	0	**65**

6.2.3 Discussion

Hand gesture recognition is a difficult problem and this experiment was only a small step towards trying to achieve the results needed in the field of real-time human computer interaction.

The main goal of the study was to learn the type of hand features that could be used for the problem at hand, and start learning how the type of selected features behave in terms of supervised machine learning. Thus, a comparative study of four machine learning algorithms applied to two datasets, composed of different types of features, for static posture recognition and classification for human computer interaction was carried out. As explained before, all the experiments were done with the RapidMiner tool, in an Intel Core i7 (2.8 GHz) Max OSX computer with 4Gb (DDR3) of RAM.

In terms of accuracy, it is possible to see from the obtained results that dataset 2 achieved a lower accuracy than dataset 1 in all the algorithms, except in the *Naïve Bayes*. On the other hand, although the best results have been obtained by the neural network with dataset 1, it took a lot of time to train and test the data as can be seen in Table 9, with a dataset comprising few records. With shorter time periods we have the k-NN algorithm with only 8 seconds used and an accuracy result very close to that obtained by the ANN, and the SVM algorithm with 3 minutes and 10 seconds and accuracy superior to 90%. For three of the learning algorithms, with dataset 1, it was possible to obtain accuracies above 90%, which was encouraging.

Although the number of records in the two databases was different, this factor does not contribute to the differences in accuracy. The number of features present in each dataset has however contributed to the time spent in the training and testing phases. Also, in dataset 1 the variance feature in the dataset is constant and equal to 1. This is due to the normalization (*z-normalization*) step taken, and because of this, this feature does not bring any added value to the final result and can be discarded in future experiments.

Some authors had already used the HoG as feature vectors for gesture classification. As seen before, this gave some problems with different gestures having similar orientation histograms. To avoid this type of problems many authors used variations of it, where for example the features are computed by dividing the image into overlapped blocks called cells. The resulting HoG is obtained by a concatenation of all the obtained histograms of the individual cells (Yafei Zhao et al. [21]). Other approaches used a skin colour histogram of gradients, which combine skin colour cues with HoG features to construct a novel feature: SCHOG. With the approach developed (feature vectors that are formed by a composition of individual features) better results than the first approach were achieved but it was inferior to the second one. Compared to other variations and other tests performed with similar features, like the one by Zondag et al**.** [20], our approach seemed very promising, and further tests should be performed, especially with the HoG operator, which being simple and fast to compute offers advantages with illumination variation images, and the radial signature, which being also simple in terms of computationally complexity gives a good representation of the object shape.

The achieved results, permit to conclude that the dataset 1 features resulted better as possible solutions for hand gesture identification, and that further experiments should be carried out with them, or even possibly try them separately. From the obtained results, it can be easily proven that feature selection and data preparation phases are important ones, especially with low-resolution images, which is the case of depth images captured with the Kinect camera.

6.3 Comparative study of seven different feature extraction algorithms

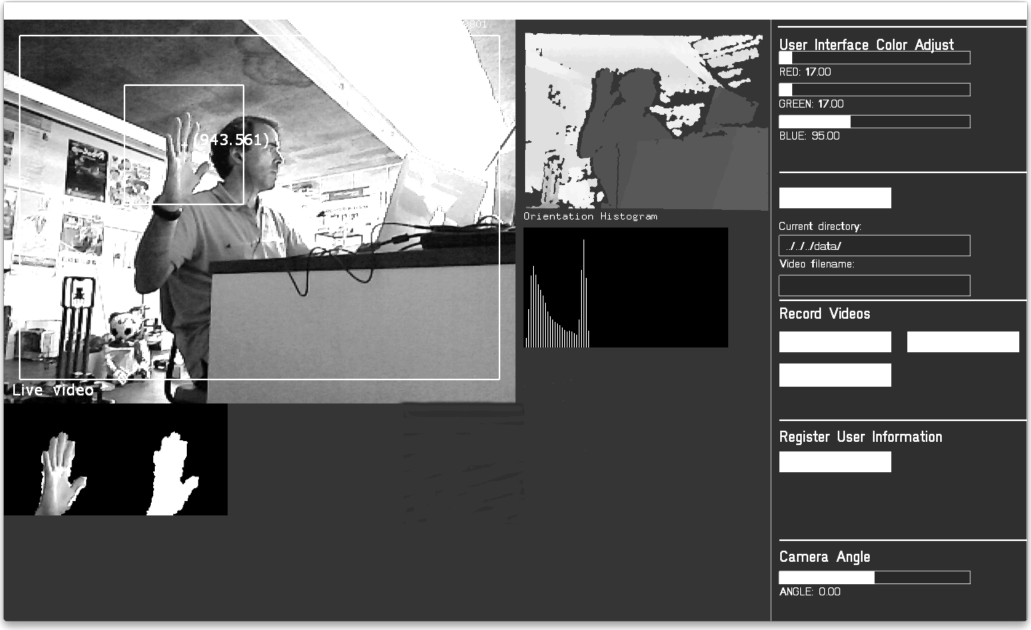
6.3.1 Experimental Setup

This experiment aimed to conduct a comparative study of seven different hand feature extraction algorithms for static hand posture classification. As explained before, careful selection of hand features for shape representation plays an important role in the final system performance. Some requirements like viewpoint invariance and user independence are important aspects to take into consideration. Also, as described in section 2.2.1, efficient shape features must present some essential properties, like for example translation, rotation and scale invariance. So, in this study it was important to understand which features, that when used isolated, responded better in real-time human-computer interaction systems and at the same time addressed the described properties being simple in terms of computationally efficiency.

For that, seven datasets with different features extracted from the segmented hand were used. The hand features used in the current experiment, and described in section 2.2, were: *the Radial Signature (RS), the Radial Signature Fourier Descriptors (RSFD), the Centroid Distance (CD), the Centroid Distance Fourier descriptors (CDFD), the Histogram of Gradients (HoG), the Shi-Tomasi Corner Detector* and the Un*iform Local Binary Patterns (ULBP)*.

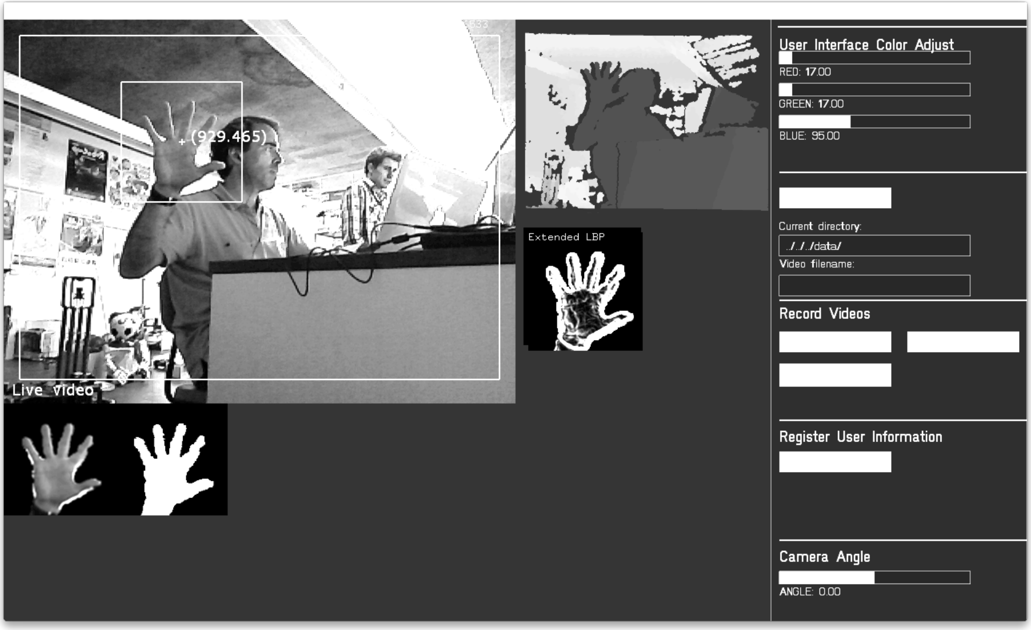
For the problem at hand, two types of images obtained with the Kinect camera were used during the feature extraction phase. The first one, the hand grey scale image was used with the HoG operator (Figure 71), the LBP (local binary pattern) operator (Figure 72) and the Shi-Tomasi corner detector (Figure 73). The second one, the binary hand blob, was used in the Radial Signature (Figure 74), the Radial Signature Fourier Descriptors (Figure 75), the Centroid Distance Signature (Figure 76) and the Centroid Distance Signature Fourier Descriptors (Figure 77).

For all the operators, as explained in section 3.2, the hand is detected and tracked as shown on each of the images, where the hand being tracked is surrounded by a white square labelled with the hand world position. For the ones that use the grey image, like the HoG operator, described in section 2.2.3, it can be seen in Figure 71 both extracted hand images below the RGB image and to the right the respective orientation histogram image. For the Local Binary Pattern operator, described in section 2.2.4, it can be seen in Figure 72 the obtained LBP image below the depth camera image.



RED: 17.00
GREEN: 17.00
BLUE: 95.00
Current directory: ../../data/
Video filename:
ANGLE: 0.00

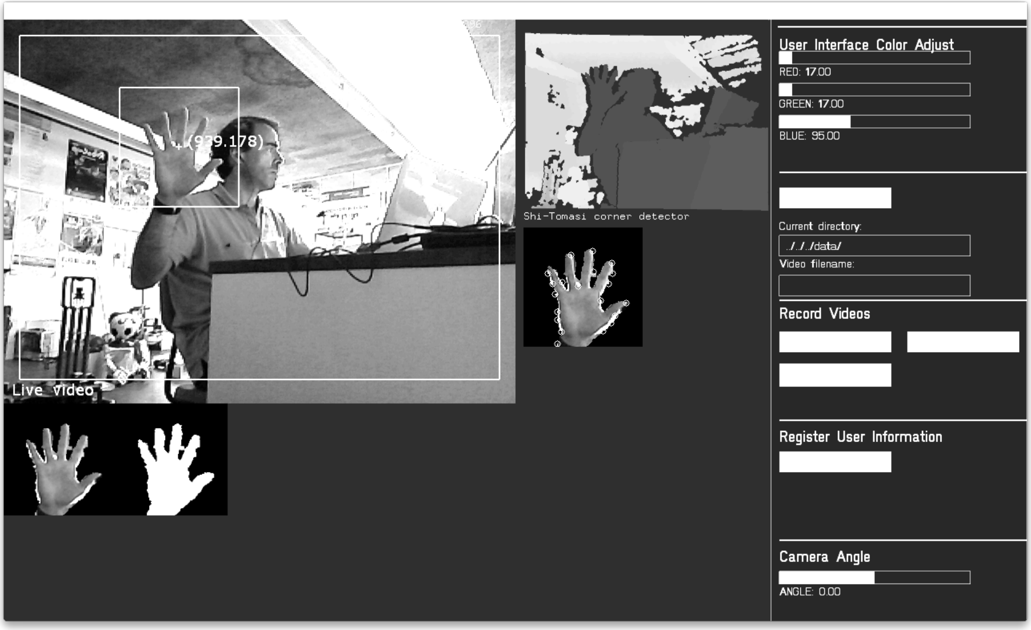
Figure 71. HoG (Histogram of Gradients) operator feature extraction.



RED: 17.00
GREEN: 17.00
BLUE: 95.00
Current directory: ../../data/
Video filename:
ANGLE: 0.00

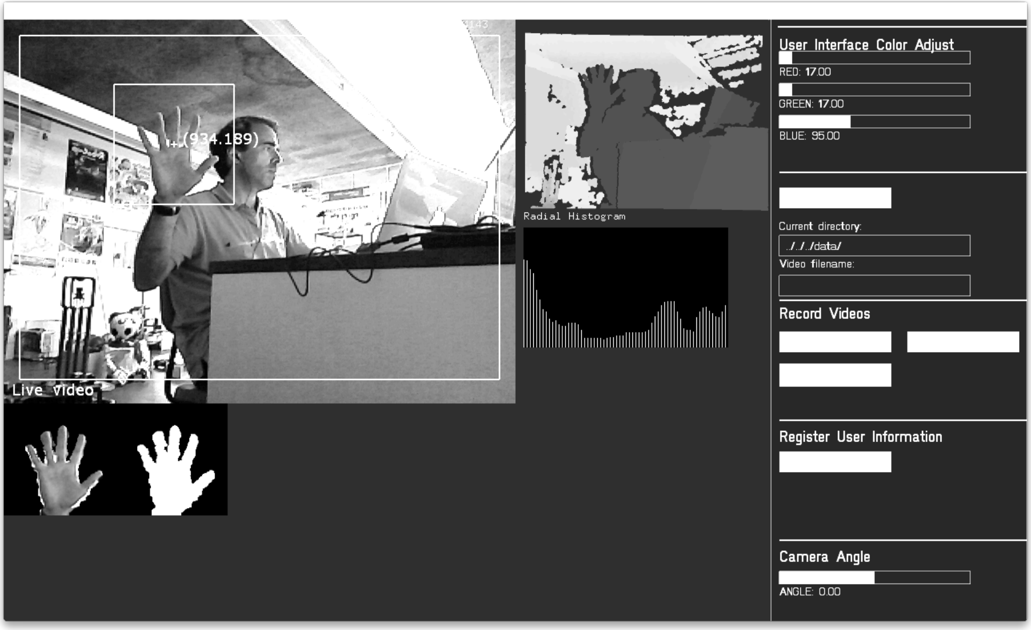
Figure 72. Local Binary Pattern operator feature extraction.

For the Shi-Tomasi corner detector, described in section 2.2.6, we can see in Figure 73 the identified points of interest displayed on top of the hand image, below the depth camera image. For those operators that used the hand binary blob, like the radial signature and the centroid distance, described in section 2.2.2, Figure 74 and Figure 76 show the respective histogram images below the depth camera image. For the Fourier Descriptors, described in section 2.2.5, the first ten obtained Fourier Descriptors are represented below the respective signature as shown in Figure 75 and Figure 77. This representation was used for control purposes only.


(939.178)
Live video
Shi-Tomasi corner detector
User Interface Color Adjust
Record Videos
Register User Information
Camera Angle

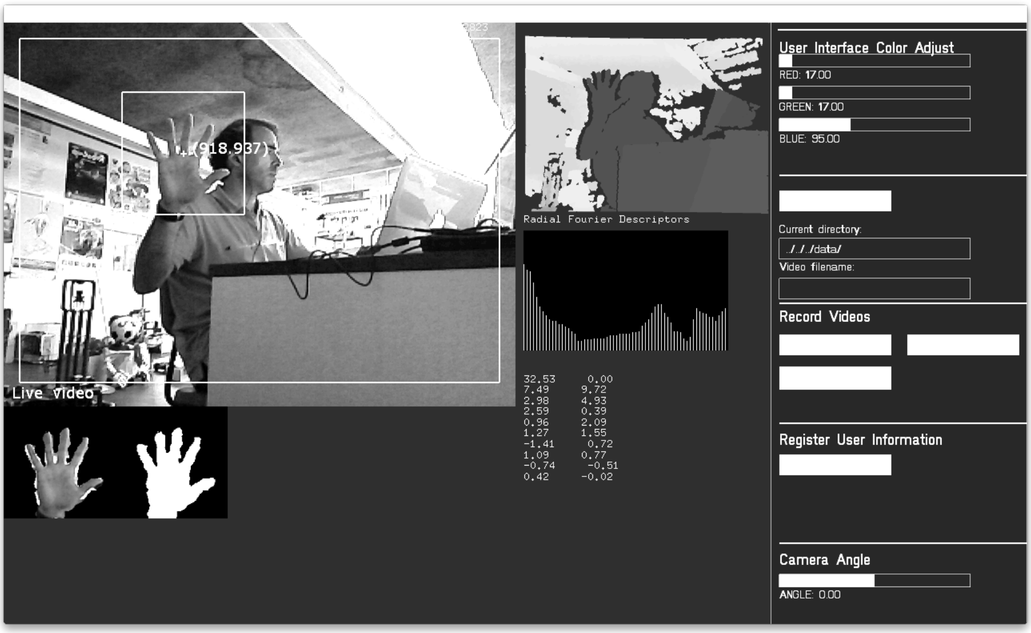
RED: 17.00
GREEN: 17.00
BLUE: 95.00
Current directory: ../../../data/
Video filename:
ANGLE: 0.00

Figure 73. The Shi-Tomasi corner detector operator feature extraction



RED: 17.00
GREEN: 17.00
BLUE: 95.00
Current directory: ../../../data/
Video filename:
ANGLE: 0.00

Figure 74. Radial Signature operator feature extraction.


(918.937)
Live video
Radial Fourier Descriptors
User Interface Color Adjust
Record Videos
Register User Information
Camera Angle

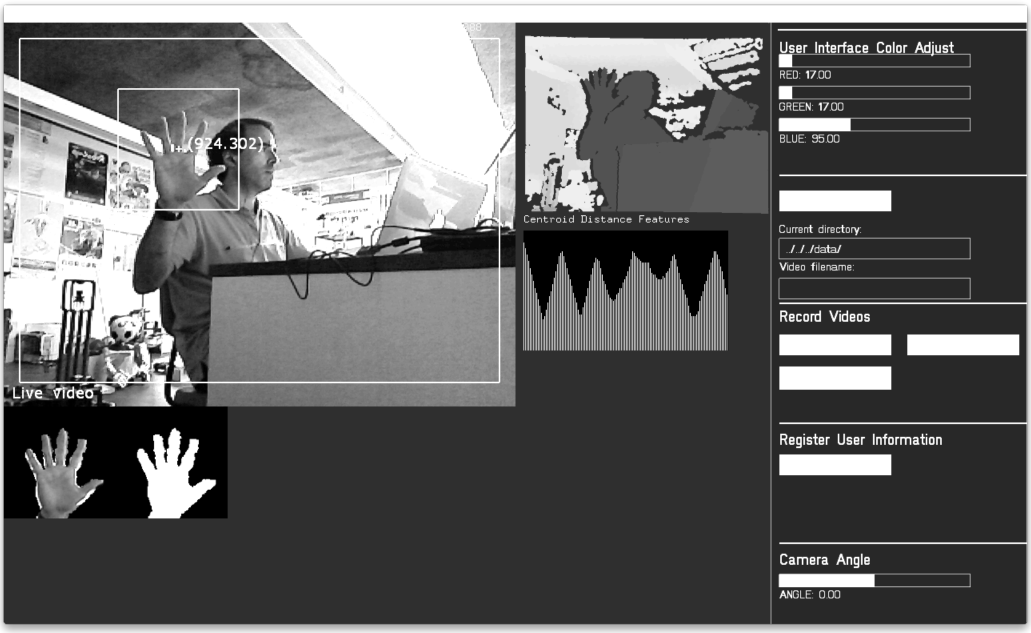
RED: 17.00
GREEN: 17.00
BLUE: 95.00

Current directory: ../../data/
Video filename:

ANGLE: 0.00

```
32.53     0.00
7.49     9.72
2.98     4.93
2.59     0.39
0.96     2.09
1.27     1.55
-1.41     0.72
1.09     0.77
-0.74     -0.51
0.42     -0.02
```

Figure 75. Radial Signature Fourier Descriptors (RSFD) operator feature extraction.

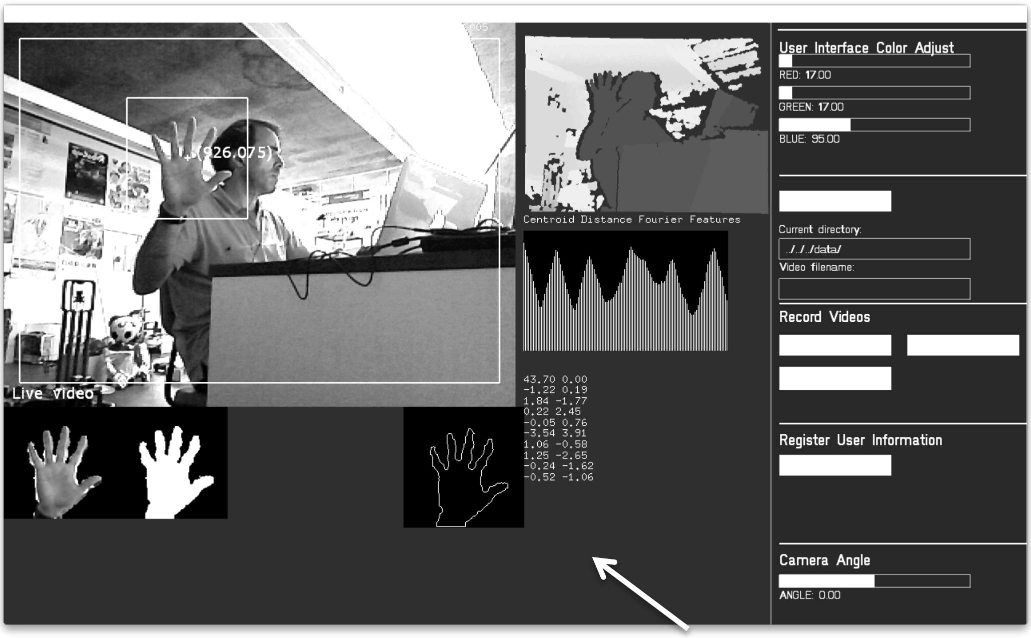

(924.302)
Live video
Centroid Distance Features
User Interface Color Adjust
Record Videos
Register User Information
Camera Angle

RED: 17.00
GREEN: 17.00
BLUE: 95.00

Current directory: ../../data/
Video filename:

ANGLE: 0.00

Figure 76. Centroid Distance operator feature extraction.


(926,075)
Live video
Centroid Distance Fourier Features
User Interface Color Adjust
Current directory:
Video filename:
Record Videos
Register User Information
Camera Angle

RED: 17.00
GREEN: 17.00
BLUE: 95.00
Current directory: ../../../data/
ANGLE: 0.00

```
43.70 0.00
-1.22 0.19
1.84 -1.77
0.22 2.45
-0.05 0.76
-3.54 3.91
1.06 -0.58
1.25 -2.65
-0.24 -1.62
-0.52 -1.06
```

Figure 77. Centroid Distance Fourier Descriptors operator feature extraction.

As stated before, the main goal of the experiment was to learn features, that isolated, responded better in real-time human-computer interaction systems. For that, a gesture vocabulary with 10 hand postures (represented in Figure 78) was defined, and videos from 20 users performing the postures, in front of the camera, for later processing were recorded. For data acquisition, an application capable of recording videos from the Kinect was implemented. The application was built in C++ with openFrameworks and the OpenNI library [131]. The main user interface is shown in Figure 79.

For each user, the videos were saved in a folder with an associated user information file (name, age, sex and gender). During gesture recording the application displays the necessary information to help the user know at any instant the current hand posture to perform and time spent in the current process, as shown in Figure 80. Each posture is recorded during 15 seconds, after which the application switches to a pause state. In this phase, the next posture to be recorded is shown on the right side of the user interface, under "*Hand Command*" and a countdown value is displayed on top of the RGB image, enabling user hand posture changing (Figure 81).

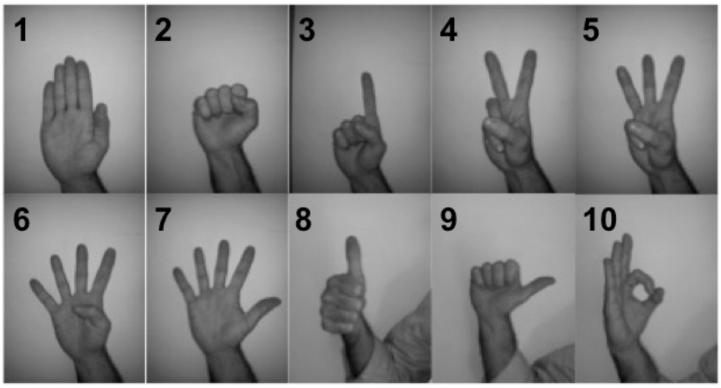
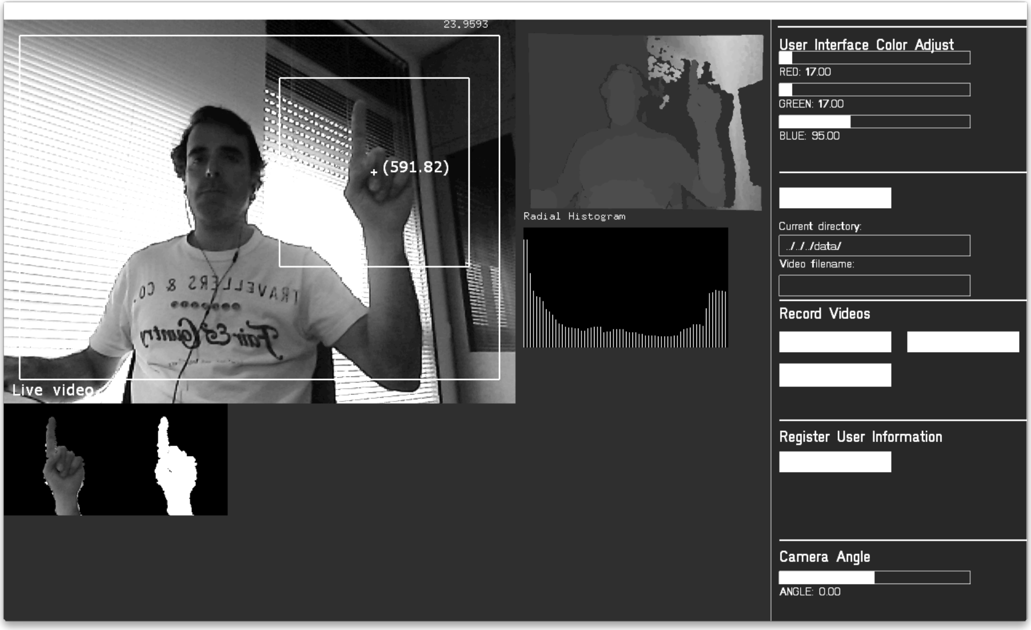


Figure 78. The defined gesture vocabulary (1. palm, 2. fist, 3. one-finger, 4. two-fingers, 5. three-fingers, 6. four-fingers, 7. five-fingers, 8. ok, 9. move-left and 10. circle)



RED: 17.00
GREEN: 17.00
BLUE: 95.00
Current directory: ../../../data/
Video filename:
ANGLE: 0.00

Figure 79. Main user interface for the Kinect video recording application.

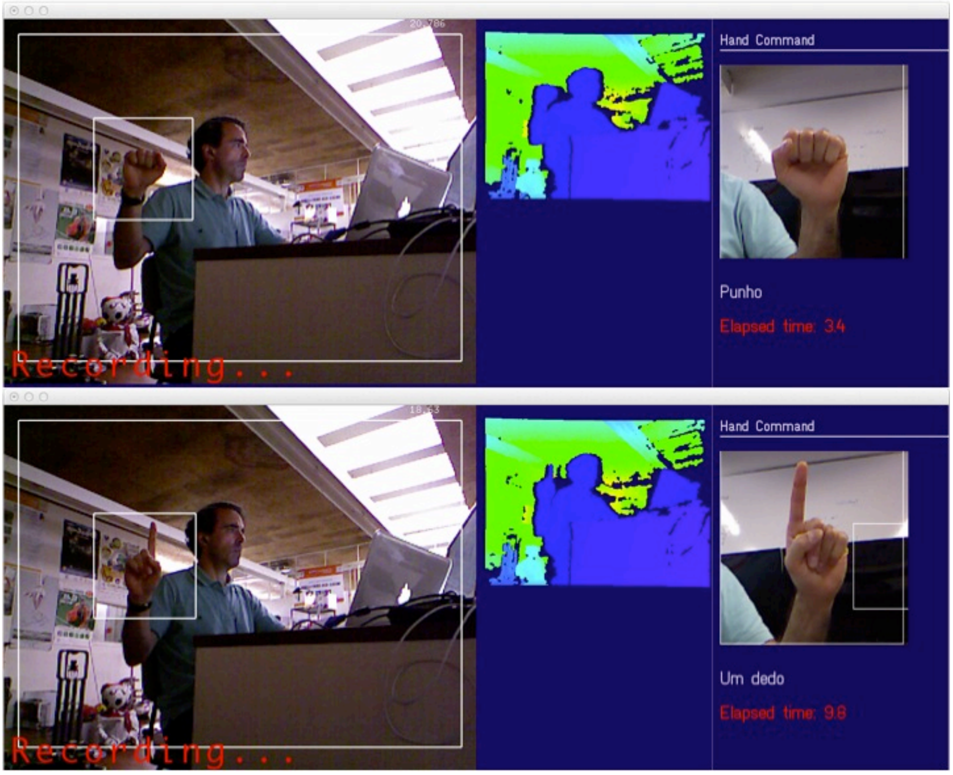


Figure 80. Posture recording. "**Fist**" and "**One Finger**" postures.

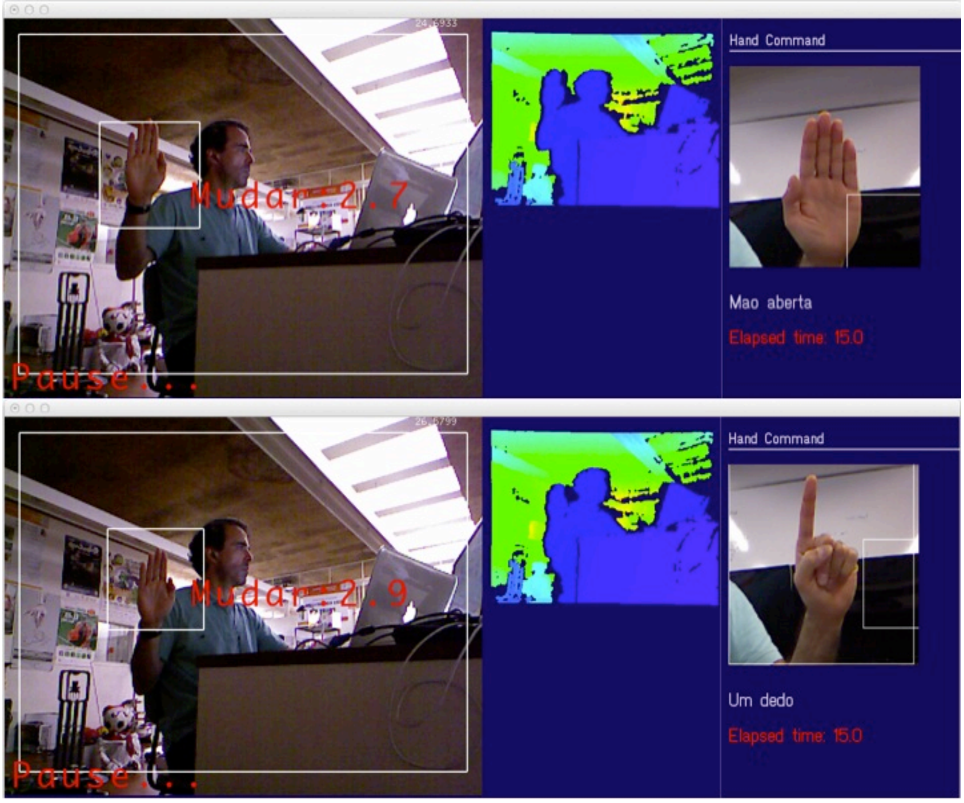
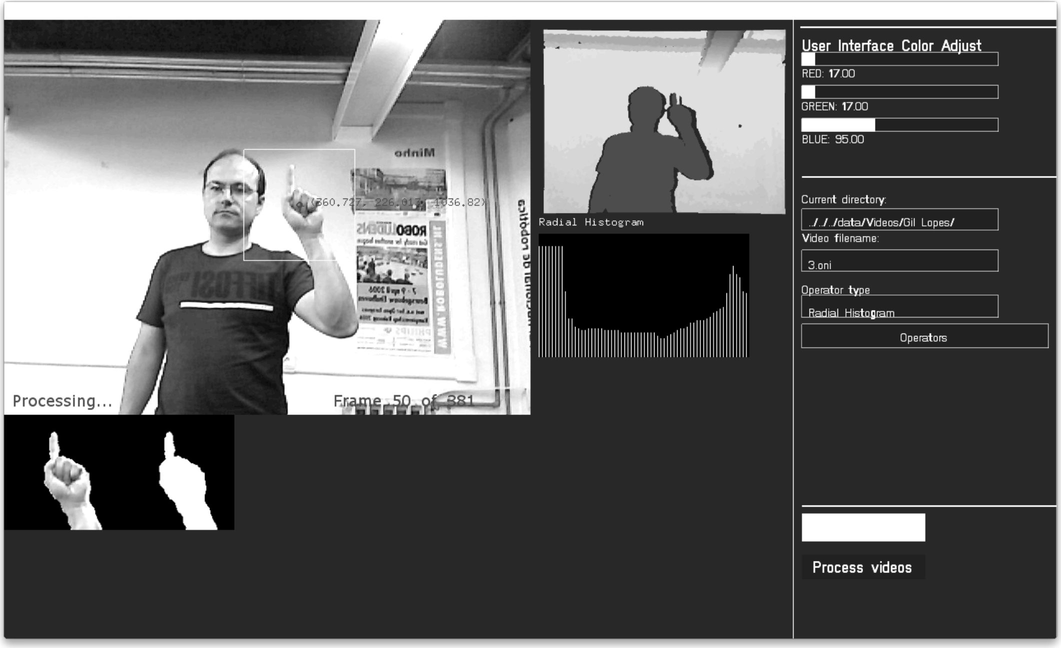


Figure 81. Posture recording pause period.

As explained before (section 3.3.1), feature selection, data set preparation and data transformation, are important phases in the process of data analysis. To construct the right model it is necessary to understand the data under analysis. Successful data mining involves far more than selecting a learning algorithm and running it over your data [77].

In order to process the recorded videos, a C++ application, using openFrameworks, OpenCV [28] and OpenNI [131], was developed. The application main user interface, shown in Figure 82, allows the processing of a single file or the selection of a user folder in order to process all the associated video files as shown in Figure 83. The application runs through all the folder video files and extracts for each one of the operators under study the respective hand features and saves them in separate datasets.

On the application user interface, information regarding the system state is displayed on top of the camera RGB image. In the image of Figure 82, for example, it is possible to see that the system is in a "*Processing*" state and is processing frame 50 of 381. The type of algorithm or operator in use at any time, for feature processing, is also an option that can be selected by the user as shown in Figure 84.



Processing...

Frame 50 of 381

RED: 17.00

GREEN: 17.00

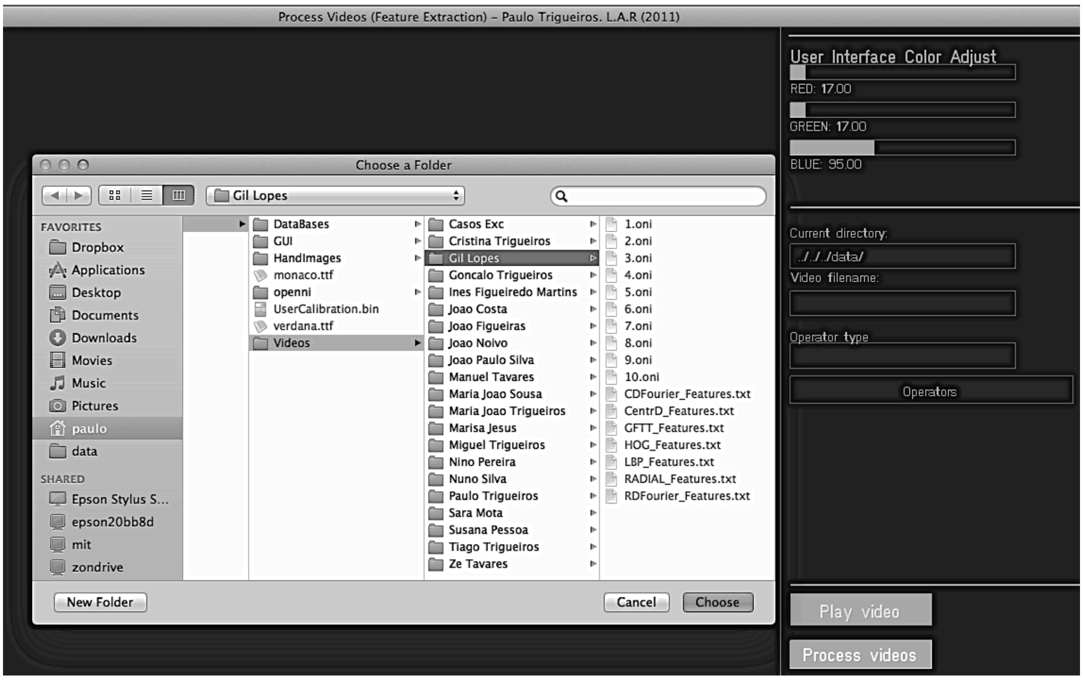
BLUE: 95.00

Current directory: ../../../data/Videos/Gil Lopes/

Video filename: 3.oni

Operator type: Radial Histogram

Figure 82. Main user interface for the videos processing application.



Process Videos (Feature Extraction) – Paulo Trigueiros. L.A.R (2011)

User Interface Color Adjust

RED: 17.00

GREEN: 17.00

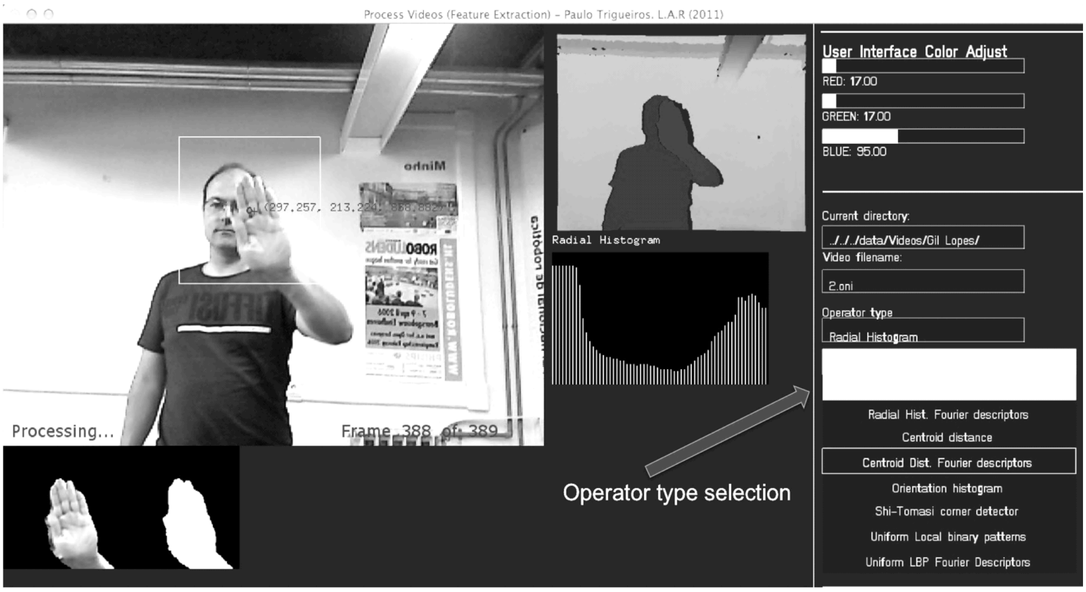
BLUE: 95.00

Current directory: ../../data/

Video filename:

Operator type

Figure 83. User data folder selection for feature extraction.



Process Videos (Feature Extraction) – Paulo Trigueiros. L.A.R (2011)

User Interface Color Adjust

RED: 17.00

GREEN: 17.00

BLUE: 95.00

Current directory: ../../data/Videos/Gil Lopes/

Video filename: 2.oni

Operator type: Radial Histogram

Processing...

Frame 388 of 389

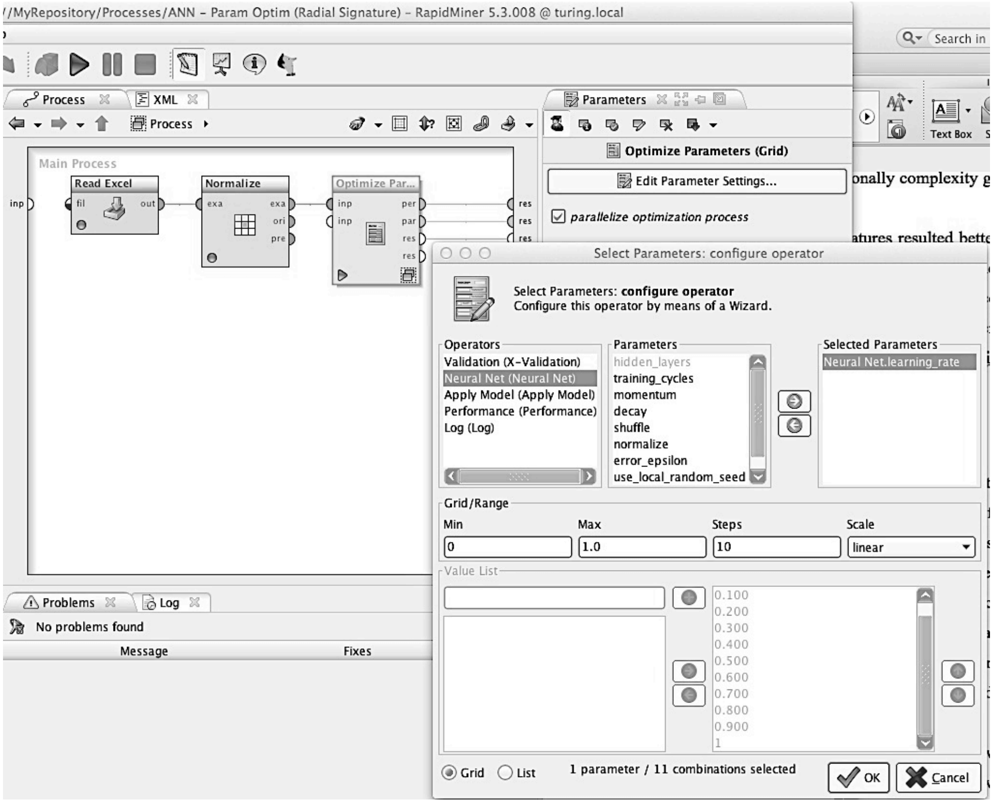
Radial Histogram
Operator type selection
Radial Hist. Fourier descriptors
Centroid distance
Centroid Dist. Fourier descriptors
Orientation histogram
Shi-Tomasi corner detector
Uniform Local binary patterns
Uniform LBP Fourier Descriptors

Figure 84. Operator type selection (in the image the Radial Histogram was selected).

All the datasets were converted to Excel files to be imported into RapidMiner for data analysis and in order to find the best learner algorithm among the following four: **k-NN**, **Naïve Bayes**, **ANN** and **SVM**.

All the datasets were analysed with RapidMiner, and as in previous examples, under the assumption of the *k-fold cross validation method* with k set to 10. The type of sampling used for the current experiments was the stratified sampling, which, as explained in the previous section builds random subsets and ensures that the class distribution in the subsets is the same as in the whole dataset, i.e. each subset contains roughly the same proportions of the number of classes. For all the algorithms under study, a parameter optimization, for some of the parameters in each algorithm, was carried out.

In the following sequence of images RapidMiner process configurations used for each of the learners is explained, with the corresponding setups for the parameter optimization. For the ANN algorithm, the number of training cycles was defined as constant and equal to 500. The learning rate was obtained via parameter optimization. The next image shows the configuration window for the optimization parameter selection where, the learning rate parameter has values defined in the range of 0 to 1 with an increment of 0.1.



/MyRepository/Processes/ANN – Param Optim (Radial Signature) – RapidMiner 5.3.008 @ turing.local

Select Parameters: configure operator

Select Parameters: **configure operator**
Configure this operator by means of a Wizard.

Operators: Validation (X-Validation), Neural Net (Neural Net), Apply Model (Apply Model), Performance (Performance), Log (Log)

Parameters: hidden_layers, training_cycles, momentum, decay, shuffle, normalize, error_epsilon, use_local_random_seed

Selected Parameters: Neural Net.learning_rate

Grid/Range: Min 0, Max 1.0, Steps 10, Scale linear

Value List: 0.100, 0.200, 0.300, 0.400, 0.500, 0.600, 0.700, 0.800, 0.900, 1

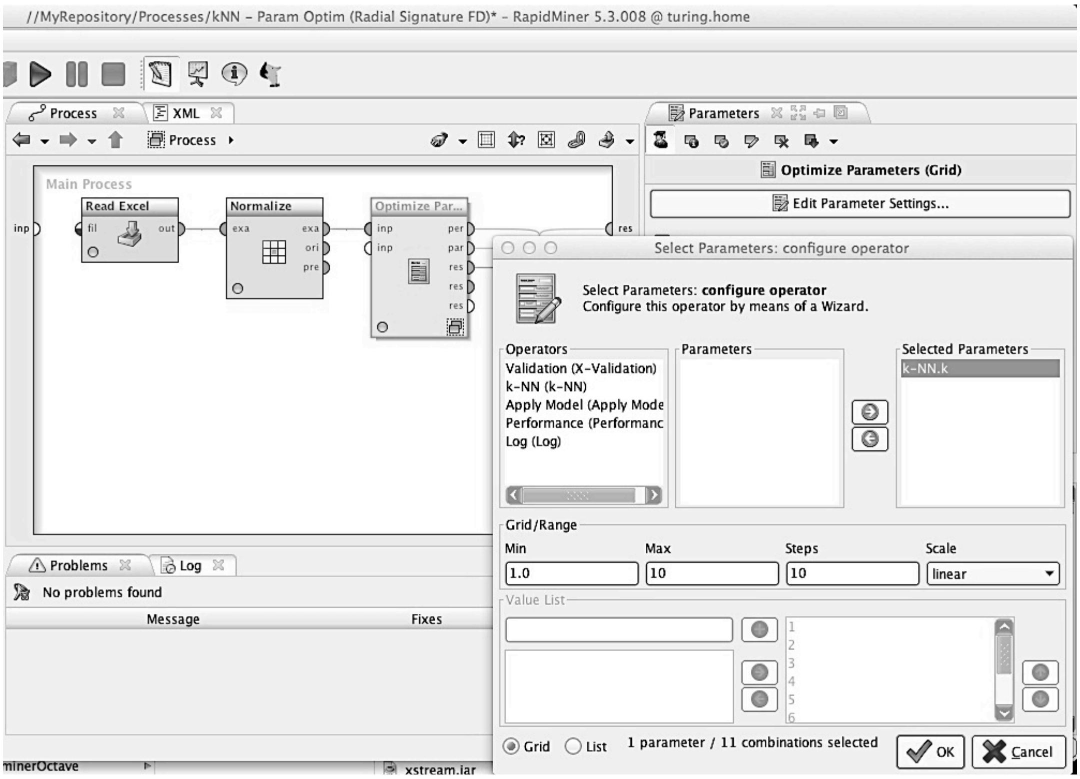
● Grid ○ List 1 parameter / 11 combinations selected

☑ parallelize optimization process

No problems found

Figure 85. RapidMiner configuration for ANN Radial Signature feature analysis, with parameter optimization.

For the k-NN the only parameter optimized was the number of neighbours, k. The following image shows the first configuration window, with values defined in the range 1 to 10 with a linear scale increment.



//MyRepository/Processes/kNN – Param Optim (Radial Signature FD)* – RapidMiner 5.3.008 @ turing.home

Select Parameters: configure operator

Select Parameters: configure operator
Configure this operator by means of a Wizard.

Operators:
Validation (X-Validation)
k-NN (k-NN)
Apply Model (Apply Mode
Performance (Performanc
Log (Log)

Selected Parameters: k-NN.k

Grid/Range
Min: 1.0
Max: 10
Steps: 10
Scale: linear

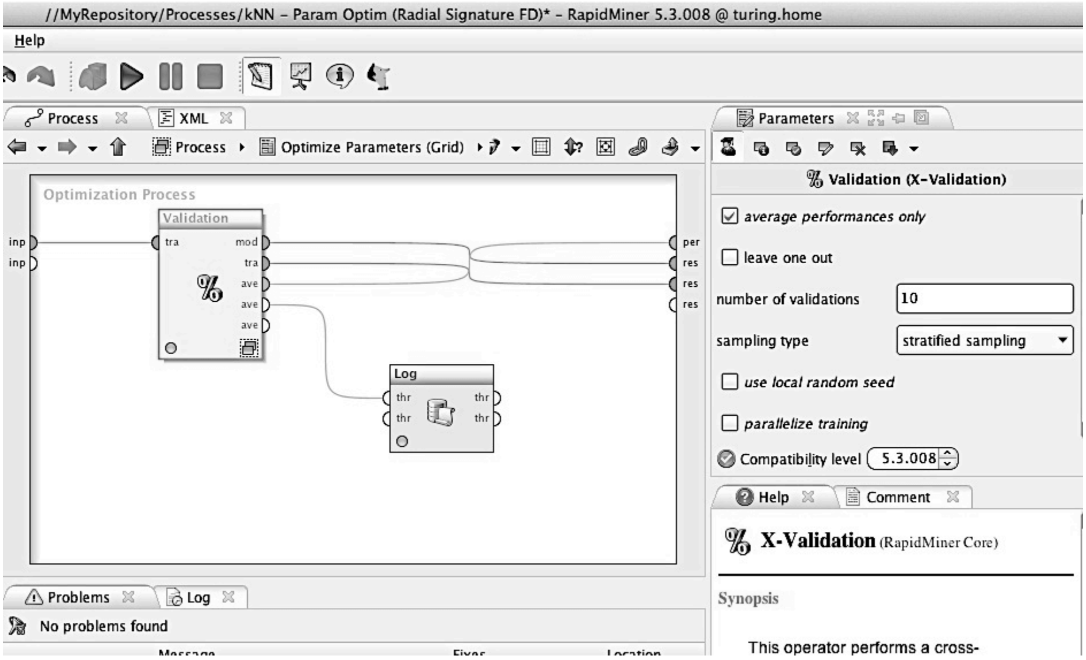
● Grid ○ List 1 parameter / 11 combinations selected

No problems found

Figure 86. RapidMiner configuration for the k-NN Radial Signature Fourier Descriptors feature analysis with parameter optimization (first screen).

In the following window, the optimization setup window, the cross validation operator is defined, and as it can be seen it has attached a '*Log'* operator. This option enables the visualization of the values obtained during the optimization process, in terms of accuracy for all the selected parameters as shown in the simple example output of Figure 88.

Figure 89 shows the training and testing configuration window, where on the right side, under "*Parameters*", it can be seen the configuration for the k-NN algorithm with k=1 and the "*Numerical measure*" set to the Euclidean distance. These values were set according to the obtained ones during the optimization phase.



//MyRepository/Processes/kNN – Param Optim (Radial Signature FD)* – RapidMiner 5.3.008 @ turing.home

Validation (X-Validation)

☑ average performances only

☐ leave one out

number of validations: 10

sampling type: stratified sampling

☐ use local random seed

☐ parallelize training

Compatibility level: 5.3.008

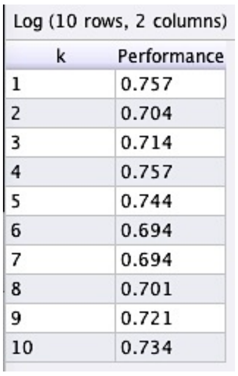
No problems found

X-Validation (RapidMiner Core)

Synopsis

This operator performs a cross-

Figure 87. Validation configuration with an option for Log display.

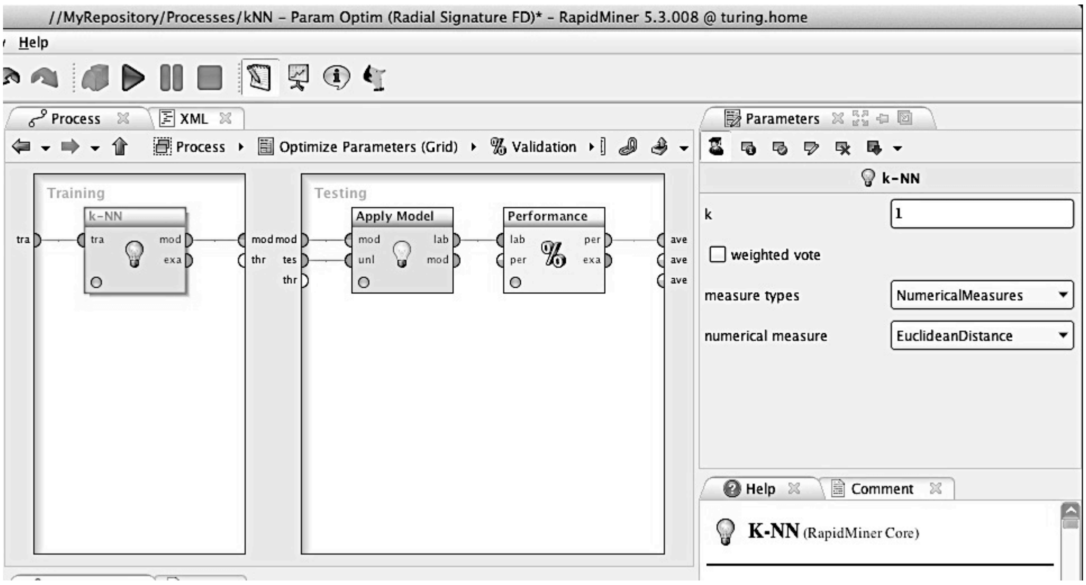


Log (10 rows, 2 columns)

k	Performance
1	0.757
2	0.704
3	0.714
4	0.757
5	0.744
6	0.694
7	0.694
8	0.701
9	0.721
10	0.734

Figure 88. Example of Log table obtained during parameter optimization.

For SVM training and testing, with the HoG features, the kernel type and the C parameter were once again obtained via parameter optimization. Figure 90 to Figure 92 show the parameter optimization configuration window, the cross-validation configuration window and the training and testing configuration window. The first one shows the range defined for the SVM parameter C to be optimized. It is defined with values in the range 0 to 10 with a linear scale. The last one shows the type of SVM used, the C-SVC, and some of the parameters that can be learned.



//MyRepository/Processes/kNN – Param Optim (Radial Signature FD)* – RapidMiner 5.3.008 @ turing.home

Process ▸ Optimize Parameters (Grid) ▸ Validation

k-NN

k: 1

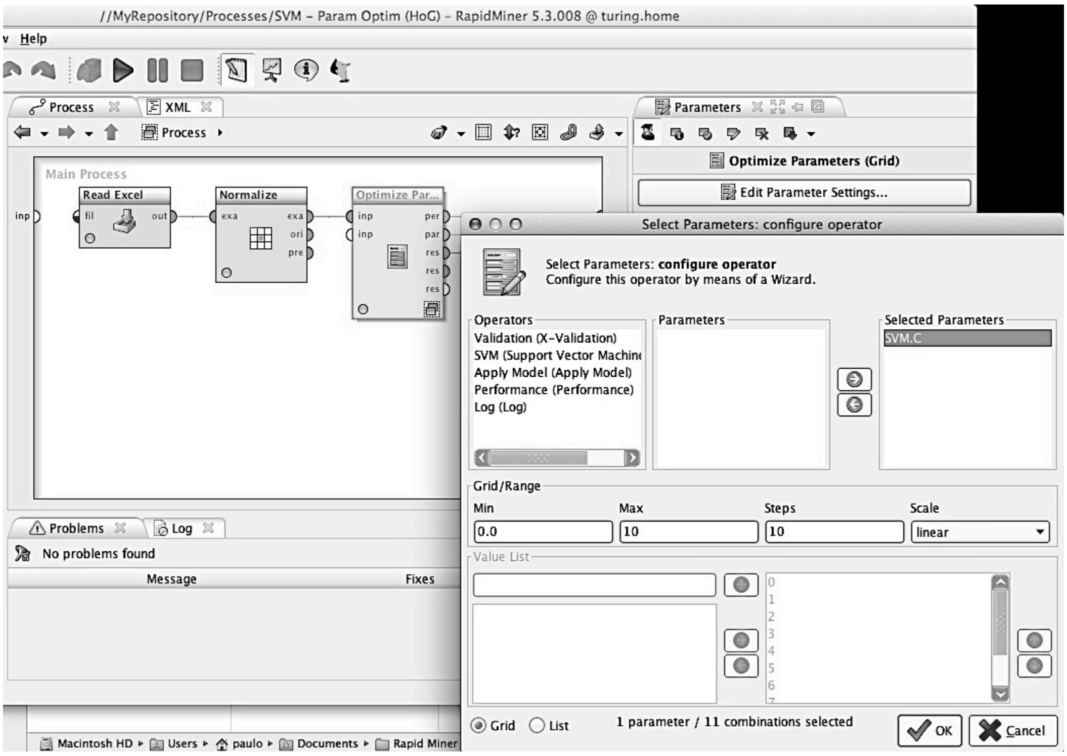
☐ weighted vote

measure types: NumericalMeasures

numerical measure: EuclideanDistance

K-NN (RapidMiner Core)

Figure 89. The training and testing configuration window for the k-NN learner.



//MyRepository/Processes/SVM – Param Optim (HoG) – RapidMiner 5.3.008 @ turing.home

Optimize Parameters (Grid)

Edit Parameter Settings...

Select Parameters: configure operator

Select Parameters: **configure operator**
Configure this operator by means of a Wizard.

Operators:
- Validation (X-Validation)
- SVM (Support Vector Machine
- Apply Model (Apply Model)
- Performance (Performance)
- Log (Log)

Selected Parameters: SVM.C

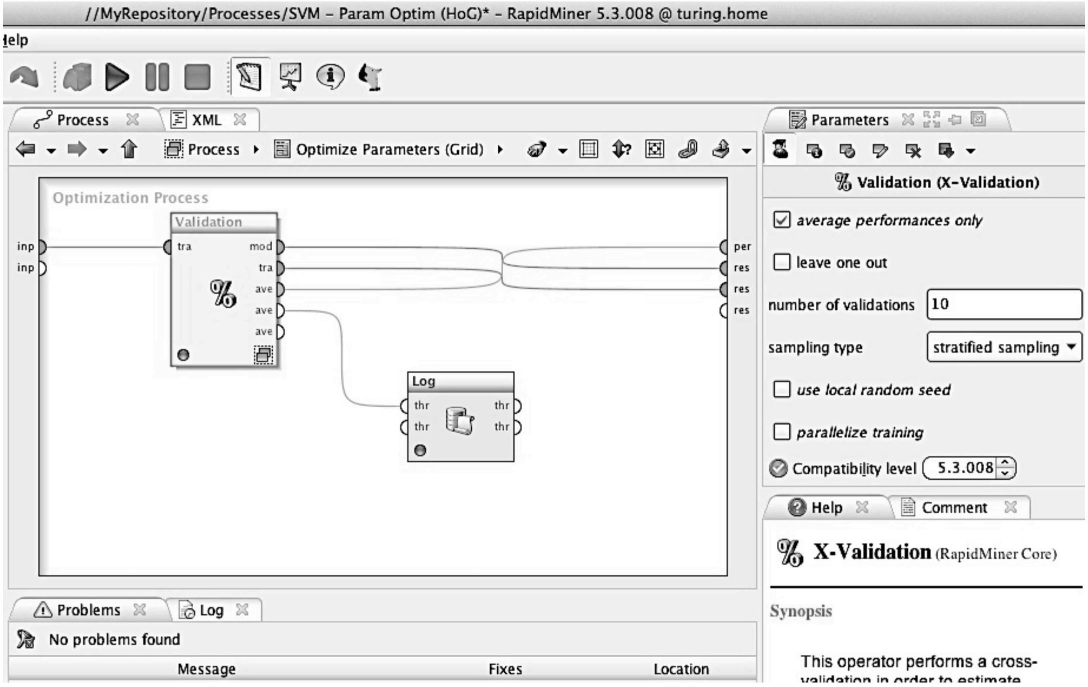
Grid/Range

Min	Max	Steps	Scale
0.0	10	10	linear

● Grid ○ List   1 parameter / 11 combinations selected

No problems found

Figure 90. RapidMiner configuration for SVM histogram of gradients feature analysis with parameter optimization (first screen).



//MyRepository/Processes/SVM – Param Optim (HoG)* – RapidMiner 5.3.008 @ turing.home

Process › Optimize Parameters (Grid)

Validation (X-Validation)

☑ average performances only

☐ leave one out

number of validations 10

sampling type stratified sampling

☐ use local random seed

☐ parallelize training

Compatibility level 5.3.008

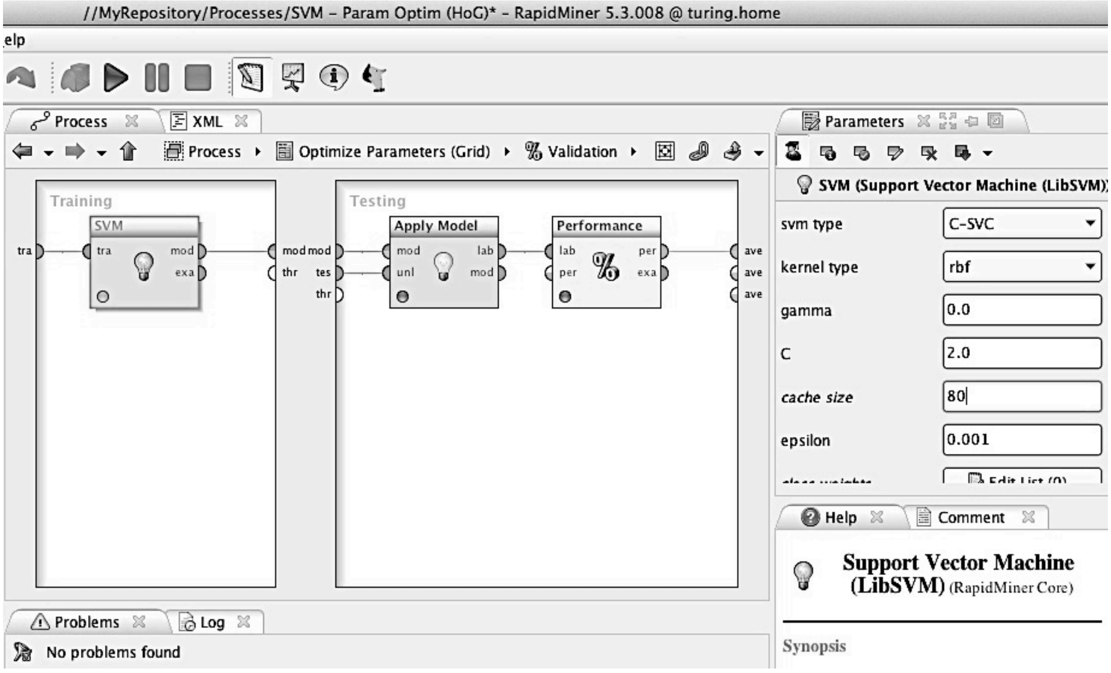
X-Validation (RapidMiner Core)

Synopsis

This operator performs a cross-validation in order to estimate

No problems found

Figure 91. RapidMiner SVM cross validation and log parameter configuration window.



//MyRepository/Processes/SVM – Param Optim (HoG)* – RapidMiner 5.3.008 @ turing.home

Process › Optimize Parameters (Grid) › Validation ›

SVM (Support Vector Machine (LibSVM))

svm type C-SVC

kernel type rbf

gamma 0.0

C 2.0

cache size 80

epsilon 0.001

Support Vector Machine (LibSVM) (RapidMiner Core)

Synopsis

No problems found

Figure 92. Configuration used for SVM model training and testing.

6.3.2 Results

After analysis of the different datasets, the obtained results were in most of the cases encouraging, although in other cases weaker than we expected.

The algorithms performance, based on the counts of test records correctly and incorrectly predicted by the model, was analysed. Table 18 summarizes the best learning algorithm for each dataset with the corresponding optimized parameters obtained and its respective achieved accuracy.

Table 18. Best learning algorithm, in terms of accuracy, for each of the datasets and parameters obtained during optimization.

Dataset	**Best learning algorithm**	**Parameters obtained by optimization**	**Accuracy**
Radial Signature	Neural Net	learning rate = 0.1	**91,0%**
Centroid Distance	Neural Net	learning rate = 0.3	**90,1%**
Radial Sign. Fourier Descriptors	k-NN	k (number of neighbours) = 1	82,3%
Centroid dist. Fourier Descriptors	k-NN	k (number of neighbours) = 1	79.5%
Uniform Local Binary Patterns	SVM (libSVM)	Kernel = RBF ; C = 6	89,3%
Histogram of Gradients	SVM (libSVM)	Kernel = RBF ; C = 2	61,5%
Shi-Tomasi corners	Neural Net	learning rate = 0.1	21,9%

In order to analyse how classification errors are distributed among classes, a confusion matrix was computed for each dataset with the learner that obtained the best result. The resulting confusion matrices are represented in the following tables.

Table 19. Radial signature dataset confusion matrix.

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	**234**	1	2	2	3	4	2	6	4	6
	2	2	**290**	8	2	2	3	1	3	0	6
	3	2	1	**273**	2	4	5	5	2	2	8
	4	1	1	4	**252**	6	3	2	4	1	0
	5	5	1	4	2	**291**	7	1	5	0	0
	6	2	1	2	5	1	**281**	8	6	2	0
	7	2	1	2	4	1	3	**290**	3	0	6
	8	2	3	5	3	2	4	0	**250**	1	5
	9	7	3	9	0	2	3	2	1	**276**	4
	10	0	8	3	4	4	2	2	2	1	**258**

Table 20. Centroid distance dataset confusion matrix.

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	343	7	2	2	5	1	2	2	6	12
	2	9	335	4	4	8	4	12	3	1	1
	3	1	2	314	5	43	0	1	3	0	5
	4	1	0	2	287	7	3	1	12	1	8
	5	2	1	1	2	309	3	8	7	0	9
	6	2	1	0	7	4	345	3	5	9	4
	7	5	4	4	4	0	4	321	1	2	2
	8	3	3	9	3	5	2	1	299	3	3
	9	2	4	6	0	7	3	3	5	308	1
	10	2	4	3	8	11	5	5	9	1	271

Table 21. Radial signature Fourier confusion matrix.

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	250	1	4	2	9	12	3	3	2	2
	2	2	275	10	6	8	3	17	8	1	17
	3	3	5	249	9	7	5	6	17	0	16
	4	7	12	11	248	8	6	7	10	1	0
	5	6	2	4	20	241	16	10	14	2	8
	6	12	3	3	2	21	245	9	4	2	2
	7	3	8	3	5	4	7	228	2	0	10
	8	3	2	13	6	7	9	12	220	1	9
	9	9	1	1	0	2	4	0	6	287	1
	10	1	3	6	0	2	1	6	5	1	232

Table 22. Centroid distance Fourier confusion matrix.

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	261	17	4	3	13	9	8	5	12	5
	2	8	258	7	8	9	4	8	10	5	6
	3	9	12	295	11	8	2	6	5	6	7
	4	6	6	8	234	7	11	6	7	7	11
	5	2	3	5	6	273	3	12	4	17	17
	6	2	5	4	6	8	290	15	1	6	17
	7	1	6	6	8	3	10	284	12	9	11
	8	9	11	6	5	6	4	3	260	4	14
	9	9	6	7	11	5	7	6	6	242	8
	10	2	6	7	4	7	15	10	15	8	237

Table 23. Local binary patterns dataset confusion matrix.

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	460	7	4	4	2	2	6	5	10	15
	2	12	499	7	7	8	9	7	3	11	7
	3	2	4	457	24	11	1	2	1	5	9
	4	9	9	12	486	30	6	0	0	8	17
	5	3	15	18	35	522	8	3	0	5	17
	6	10	14	2	4	11	531	4	2	7	15
	7	3	2	1	0	0	1	517	1	2	4
	8	10	1	1	0	0	5	3	554	1	0
	9	5	7	7	8	1	9	4	0	525	4
	10	15	5	31	13	9	4	2	0	1	457

Table 24. Histogram of gradients dataset confusion matrix.

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	174	15	14	11	5	0	1	17	19	17
	2	24	207	8	11	10	13	8	9	25	12
	3	18	10	199	25	12	4	2	6	20	13
	4	7	5	22	168	24	15	3	4	10	24
	5	8	7	11	18	181	19	15	4	6	19
	6	0	7	2	9	24	195	19	5	7	15
	7	16	39	16	21	34	62	259	39	20	38
	8	10	4	3	5	6	2	1	189	5	3
	9	30	19	17	9	8	3	1	12	176	14
	10	10	15	16	23	13	11	11	3	19	161

Table 25. Shi-Tomasi corner detector confusion matrix.

		Actual class									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Predicted class	**1**	45	26	36	22	23	19	15	17	30	25
	2	22	77	34	16	10	18	16	5	15	59
	3	46	40	49	52	49	40	27	33	25	42
	4	28	30	41	43	48	35	27	23	22	27
	5	23	16	36	36	30	42	22	23	14	11
	6	29	21	39	42	46	56	53	26	29	27
	7	16	23	15	30	34	35	75	16	31	28
	8	27	5	37	23	32	37	16	139	14	9
	9	27	22	12	13	20	26	38	7	104	24
	10	24	58	21	26	23	17	27	7	30	63

6.3.3 Discussion

The goal of this experiment was to carry a comparative study of seven different algorithms for hand feature extraction, aimed at static hand gesture classification and recognition, for human computer interaction.

All the data analysis experiments were carried with RapidMiner, on an Intel Core i7 (2.8 GHz) Max OSX computer with 4Gb (DDR3) of RAM.

It was important to test the robustness of all the algorithms, applied individually, in terms of scale, translation and rotation invariance. After all the tests and having analysed the obtained results the conclusion was that further pre-processing on the video frames was necessary in order to minimize the number of different feature values obtained for the same hand posture, due to the presence of noise in the depth image. The depth video images obtained with the Kinect camera had low resolution and some noise. It was observed that some imprecision on data recordings results from those problems, leading to more difficult gesture classification and model learning. This is possible to observe on the confusion matrices, where many wrong predictions in each type of features contributed to the final low classification results, being the worst case the Shi-Tomasi corner detector.

Due to this situation, it was decided that a temporal filtering and/or a spatial filtering should be used and would be tested and analysed to see if better results could be achieved.

It has been found that the radial signature and the centroid distance were the best shape descriptors tested in this experiment in terms of robustness and computation complexity. The simplicity of these results follows the *Occam's razor* principle which states that "*simpler explanations are more plausible and any unnecessary complexity should be shaved off*".

Better results were expected from the Fourier descriptors, after having analysed related work on the area with this type of features. In this case, even with the algorithm that achieved the best results, the k-NN with k=1, the results fell far short from the expected.

For the case of the Local Binary Patterns although the obtained accuracy was 89,3% with a SVM, some more experiences should be done in the future in order to try to achieve better results. Some authors used combinations of LBP features, like geometric moments used by Marek Hrúz et al. [51] with a combined accuracy of 99,7% for signer dependent tests but with only 57,4% for signer independent tests. Others, like Jinbing Gao et al. [53], used a combination of LBP features with HoG features to train a SVM classifier and were able to achieve an accuracy of 95,2% in real-time situations.

For the Histogram of Gradients, although with a different implementation from the ones described in the literature, the values obtained were largely disappointing with an accuracy of 61,5%. When compared with previous experiences with this type of feature, the results were much worse than expected, which has somehow demonstrated that as described in the literature, variations in its simplest form or combined with other types of features can give better results.

This experience was an important step towards starting to realize what kind of features and which classifiers could be used in future system implementations in order to obtain robust hand recognition in real-time.

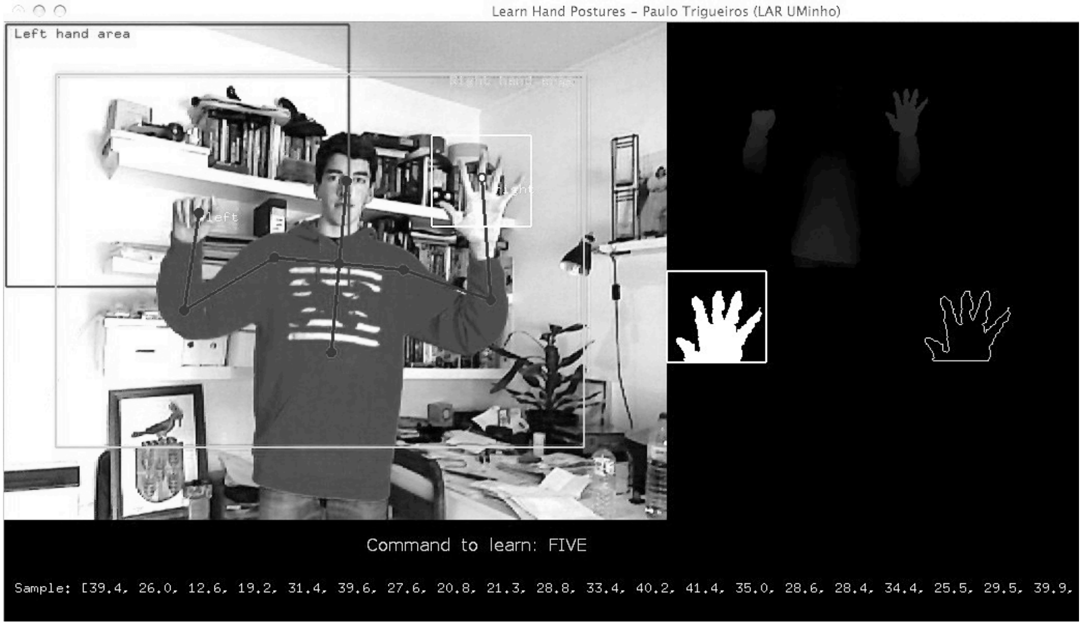
6.4 Integration of Static and Dynamic Gestures

6.4.1 Experimental Setup

This experiment main goal was to test the integration of static and dynamic gestures into a unique vision-based system for real-time human/computer interaction. For that, a decision had to be made for the type of features to use in static hand posture classification, and the type of features to use in dynamic gesture classification. Also, a method to integrate the two types of gestures in the final system had to be chosen as well.

Thus, the experiment was divided into three parts, with the first two having its own application for feature extraction and model training and testing.

For static hand posture classification it was decided to use as hand features, the centroid distance features, since these were the one that gave better results in previous experiments, being at the same simple in terms of computationally complexity, making them good candidates for applications that implied real-time recognition. So, the first part of the experiment implied the development of an application for hand feature extraction and model training and testing for a set of predefined static postures. For the type of features used, the hand binary blob had to be segmented and the hand contour calculated. For feature extraction, model learning and testing, a C++ application was built with openFrameworks, OpenCV [28], OpenNI [131] and the Dlib machine-learning library [145]. Figure 93 shows the main user interface for the application, with a sample vector (feature vector) for the posture being learned displayed below the RGB image.



```
Command to learn: FIVE

Sample: [39.4, 26.0, 12.6, 19.2, 31.4, 39.6, 27.6, 20.8, 21.3, 28.8, 33.4, 40.2, 41.4, 35.0, 28.6, 28.4, 34.4, 25.5, 29.5, 39.9,
```

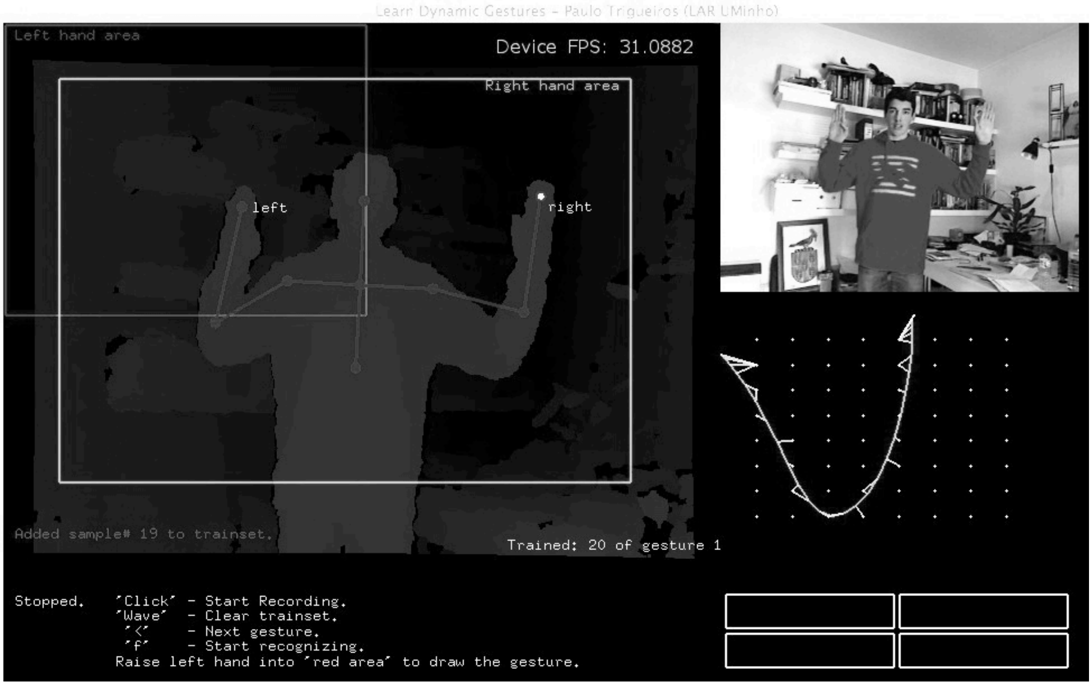
Figure 93. Static gesture feature extraction and model learning user interface

A centroid distance dataset was built with 7848 records each with 32 feature values obtained from four users. The features thus obtained were analysed with RapidMiner in order to find the best learner and the best parameters through a parameter optimization process.

The second part of the experiment implied the development of an application for the acquisition of hand motion sequences (dynamic gestures) for each of the defined gestures, feature extraction, model training and testing. For these features, the hand path set of points were used and labelled according to a set of predefined centroids or alphabet. As explained in section 3.4, gestures are time-varying processes which show statistical variations, so, it was decided to model this type of gestures with Hidden Markov Models.

A C++ application was built with openFrameworks, OpenNI and an openFrameworks addon implementation of the HMM algorithm for classification and recognition of numeric sequences (ofxSequence). This addon is a C++ porting implementation of a MATLAB code from Kevin Murphy [166]. Figure 94 shows the main user interface for the application, with a hand path drawn on top of the centroids with the corresponding labels marked as white lines. For each gesture that required training, a dataset was built and the system trained in order to learn the corresponding model parameters. For the experiment, the number of observation symbols defined was 64 with 4 hidden states. Several values for the number of observations in the set {16, 25, 36, 49, 64, 81}, and hidden states, ranging from 2 to 12 were tried out, without significant improvements for values greater than the selected ones, so those were the ones implemented in the final system.

For the tests, a new set of datasets were built with data from four different users with a total of 25 per gesture and per user, totalling 1100 records for the predefined 11 gestures (Figure 47). The final test datasets were then analysed with the previous obtained models.



```
Added sample# 19 to trainset.
Trained: 20 of gesture 1
Stopped.   'Click'  - Start Recording.
           'Wave'   - Clear trainset.
            '<'     - Next gesture.
            'f'     - Start recognizing.
           Raise left hand into 'red area' to draw the gesture.
```

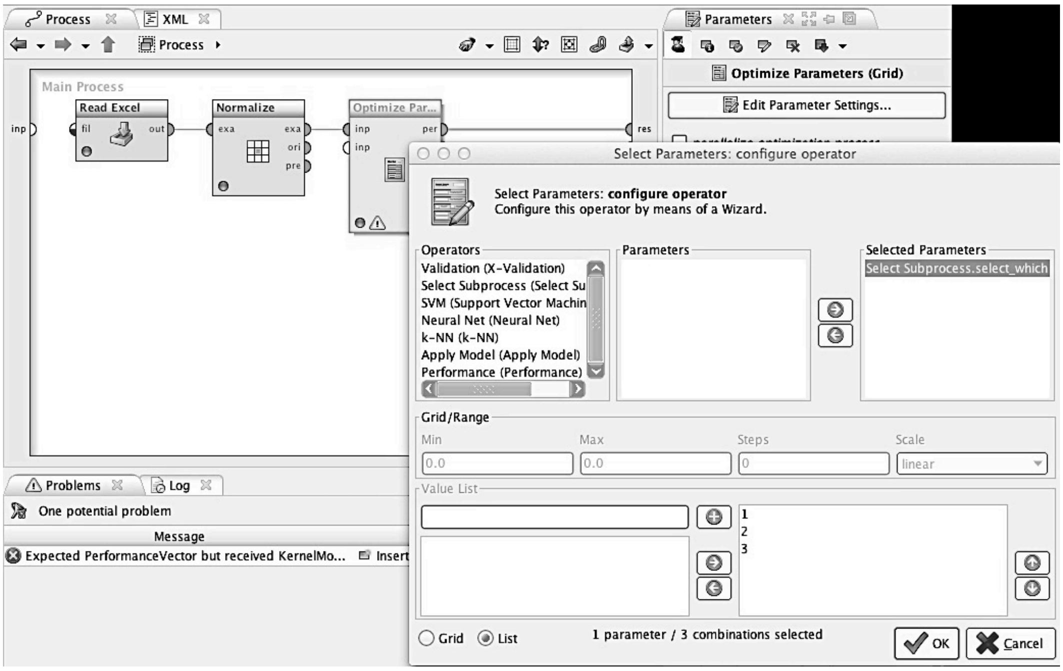
Figure 94. Dynamic gestures feature extraction and model training user interface.

The final step was concerned with the decision of how to integrate the two types of gestures. Due to the characteristics of the camera used, it was not possible to define gestures with changing postures during the all hand path, since the obtained image became blurred and therefore difficult to work. Also, as the identification of the start and end of a dynamic gesture, as in natural human language, it is still a difficult task where there is still a lot of work to be carried out, it was decided to model gestures as sequences of commands, composed of static and dynamic gestures, with a finite state machine to control the transitions between them. Although there may be other possibilities of implementation for this type of problem, this was the final decision.

6.4.2 Results

The first part of the experiment, as explained in the previous section, implied the analysis of the obtained features in order to find the best learner for the given data, and the best parameters through a parameter optimization process.

In order to find the best learner for the data under analysis, a process was built in Rapid Miner that iterates through the following three possible learning algorithms: the SVM, the ANN and the k-NN, and select the best result. The following images show the Rapid Miner configurations for the process setup, i.e., from the main screen setup with a parameter optimization definition, through the specification of the three learning algorithms to test, in the final one. By opening the "*Optimize Parameters Grid*" object by double-clicking it, we get to the Validation setup (Figure 96). Here, the parameters and type of cross-validation to be used are defined, which in the current experiment were performed with a *10-fold cross validation* with a stratified sampling type. The same way, by opening the "*Validation*" object, goes next to the Training and Testing setup (Figure 97), where is selected in the training phase a Select Sub-Process object for learner selection and in the testing phase the Apply Model object and the Performance object that will give the accuracy for each one of the chosen algorithms. Selecting the "*Select Sub-Process*" object opens the final setup window shown in Figure 98. This is where the learners to test are defined in order to obtain for the given data, the best one.



Select Parameters: configure operator

Select Parameters: **configure operator**
Configure this operator by means of a Wizard.

Operators:
- Validation (X-Validation)
- Select Subprocess (Select Su
- SVM (Support Vector Machin
- Neural Net (Neural Net)
- k-NN (k-NN)
- Apply Model (Apply Model)
- Performance (Performance)

Selected Parameters: Select Subprocess.select_which

Min 0.0, Max 0.0, Steps 0, Scale linear

Value List: 1, 2, 3

○ Grid ● List

1 parameter / 3 combinations selected

One potential problem

Expected PerformanceVector but received KernelMo...

Figure 95. First Rapid miner setup screen for the Find Best Learner Process (Read dataset and Parameter Optimization)

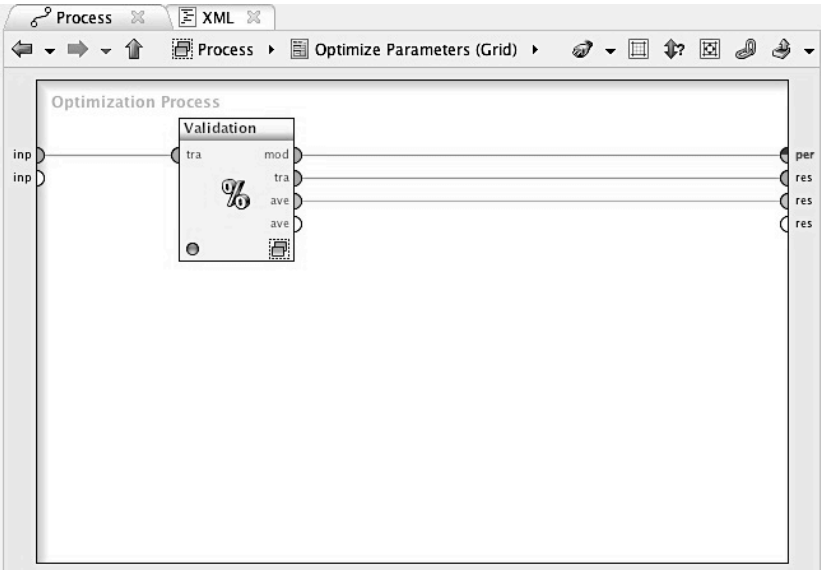
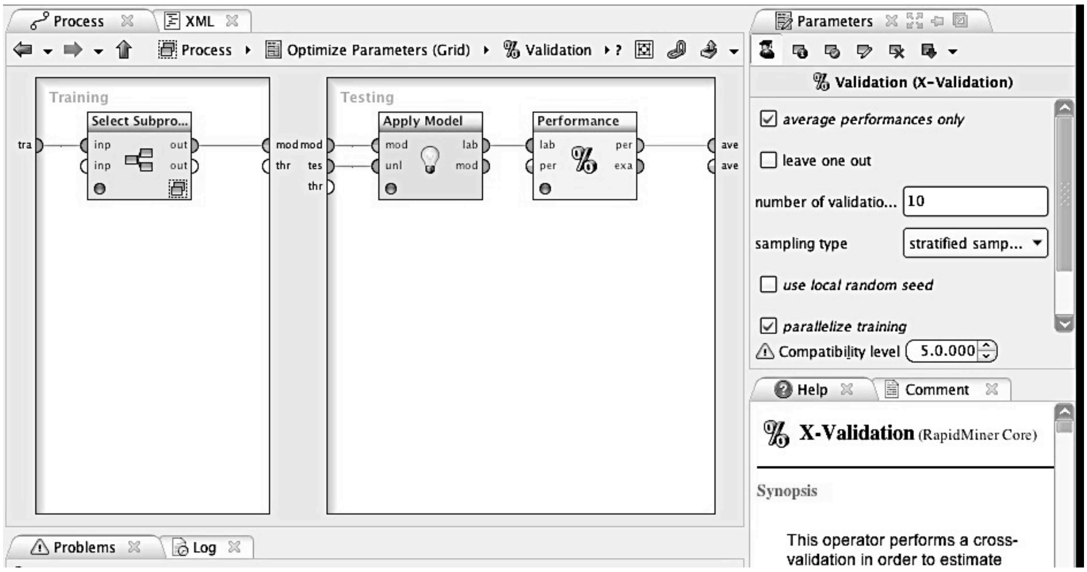


Figure 96. Second Rapid miner setup screen for the Find Best Learner Process (Validation)



Validation (X-Validation)

☑ average performances only

☐ leave one out

number of validatio... 10

sampling type stratified samp...

☐ use local random seed

☑ parallelize training

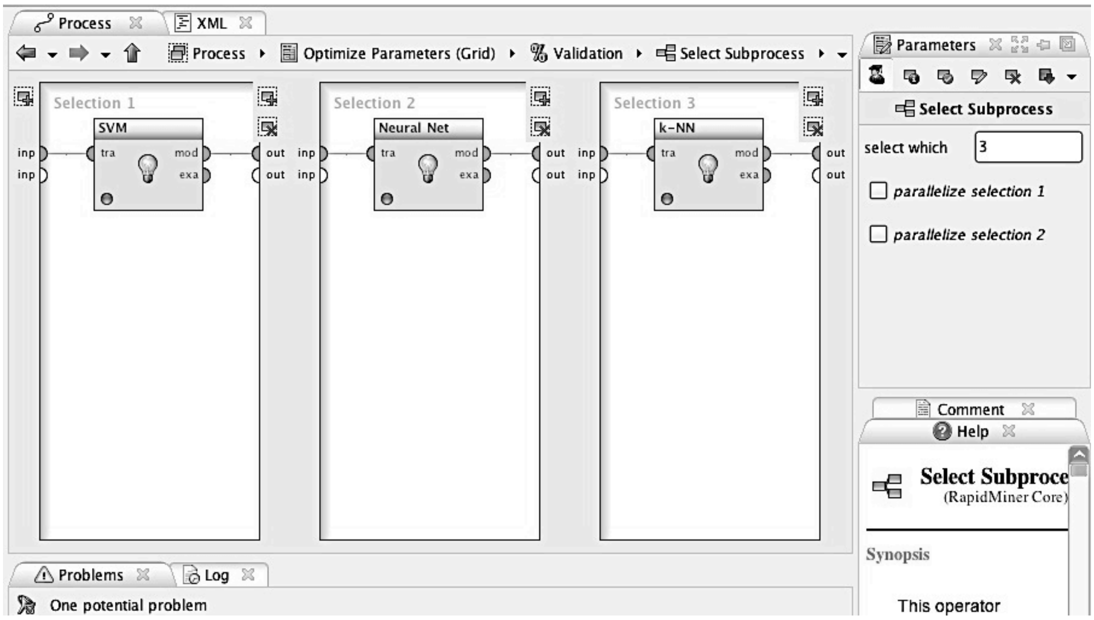
Compatibility level 5.0.000

X-Validation (RapidMiner Core)

Synopsis

This operator performs a cross-validation in order to estimate

Figure 97. Third Rapid miner setup screen for the *Find Best Learner Process* (Select Sub-Process).



Select Subprocess

select which 3

☐ parallelize selection 1

☐ parallelize selection 2

Select Subproce (RapidMiner Core)

Synopsis

This operator

One potential problem

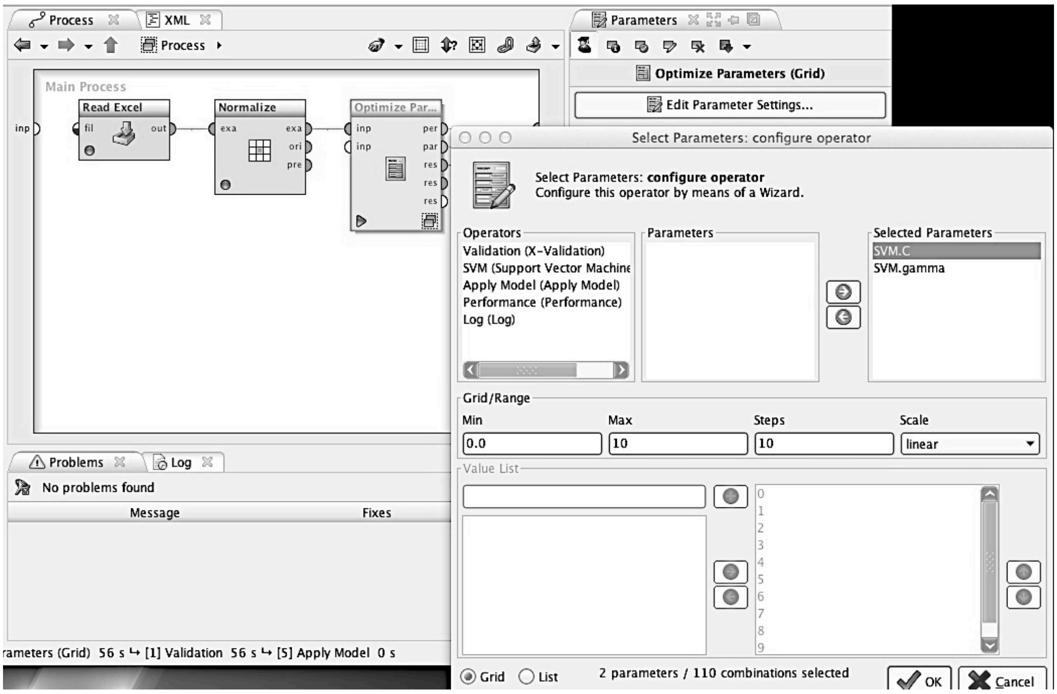
Figure 98. Fourth Rapid miner setup screen for the *Find Best Learner Process* (Learn algorithms definitions).

After running this Rapid Miner process, the best learning algorithm obtained was the SVM and therefore the one implemented in the final system.

The next step was to find the best parameters for the selected learner. For this, a *rbf kernel* was selected, since as explained in the Rapid Miner documentation, this is a reasonable first choice, and so was the one decided to use on the experiments.

A parameter optimization was thus carried out on two of the SVM parameters: the cost parameter C value, with values in the range 0 to 10, and the parameter gamma, with ranges from 0.1 to 0.9, as shown in the configuration window of Figure 99.

With the experiment an accuracy of 99,2% was achieved for the given hand features and the obtained best parameters were 6 for the cost value C and 0.1 for the gamma value with the rbf kernel.



Select Parameters: configure operator

Select Parameters: **configure operator**
Configure this operator by means of a Wizard.

Selected Parameters: SVM.C, SVM.gamma

Grid/Range: Min 0.0, Max 10, Steps 10, Scale linear

Value List: 0 1 2 3 4 5 6 7 8 9

○ Grid ○ List 2 parameters / 110 combinations selected

No problems found

rameters (Grid) 56 s ↳ [1] Validation 56 s ↳ [5] Apply Model 0 s

Figure 99. Centroid distance dataset SVM parameter optimization.

In order to analyse how classification errors were distributed among classes, a confusion matrix was computed for the dataset under analysis with the final result shown in the following table.

Table 26. Centroid distance confusion matrix

		Actual class						
		1	**2**	**3**	**4**	**5**	**6**	**7**
Predicted class	**1**	**588**	0	0	0	0	2	0
	2	2	**706**	0	0	1	0	1
	3	0	1	**578**	1	0	0	0
	4	0	0	12	**715**	3	0	0
	5	0	1	1	13	**536**	1	3
	6	1	8	0	1	5	**693**	12
	7	2	0	0	2	6	9	**751**

For the dynamic gestures testing and since RapidMiner did not had at the time of this writing the possibility to test datasets composed of HMM observation records, a function based on the following formula was implemented and included in the system in order to obtain an accuracy for each model and also a final average accuracy.

$$accuracy = \frac{\#\ correctly\ predicted\ class}{\#\ total\ testing\ class} \times 100\% \tag{31}$$

As explained in the previous section, the test datasets were analysed with the previous obtained models and the final accuracy results obtained are represented in the following table.

Table 27. Hidden Markov Models accuracy for each gesture defined (Figure 47)

Gesture	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Accuracy	75%	100%	100%	100%	92%	88%	92%	100%	100%	96%	88%

So, for the dynamic gesture recognition, with the HMM models trained with the selected features, an average accuracy of 93,72% was achieved with the test dataset.

6.4.3 Discussion

The goal of this experiment was to test the integration of static and dynamic gestures into a unique vision-based system for real-time human/computer interaction.

The experimental results showed that the system was able to recognize the combination of gestures and hand postures in real-time, although with some limitations due to the nature of the camera used. Despite these limitations, for the hand posture recognition, with the SVM model trained with the selected features, an accuracy of 99,2% was achieved.

In the centroid distance confusion matrix one can verify the existence of some high error classification values between command number four and three, between number five and four and between number six and seven, and that contributed to the 0,8% of false positives.

However, when comparing the results with those that have been reviewed in the literature in terms of SVM classification, these features managed to achieve a level of performance far superior to those presented, and so, this could be a good solution for vision-based interfaces for human / computer interaction.

Ching-Tang Hsieh et al. [93] used Fourier Descriptors as hand features to train a SVM classifier and obtained an accuracy of 93,4%. Yen-Ting Chen et al. [96] on the other hand trained three SVM's, with data from three cameras, fused later according to three different plans. With this implementation their were able to achieve a final accuracy of 93,3% although with the advantage of being a multi-view hand gesture recognition. Jinbing Gao et al. [53] proposed an adaptive HoG-LBP detector method with which they were able to achieve a final accuracy of 95,2%. Liu Yun t al. [97] presented an automatic hand gesture recognition system based on Hu invariant moment features and a SVM classifier. They were able to achieve a total recognition rate of 96,2% for a dataset composed of three postures. Nasser H. Dardas et al. [99] used SIFT features with a final accuracy of 96,23%.

For the dynamic gesture recognition, although the obtained average accuracy of 93,72% is considered satisfactory, there is still some work to be carried out in this area in order to improve the obtained results. One could observe from the individual gestures accuracy that gesture one obtained a result considered low relative to expectable. Also notice during the tests, that this particular gesture was sometimes confused with gesture number ten (END GAME). Also, gestures six and eleven obtained low accuracy values. Some more tests must be performed, including new model construction with data obtained from more users and features obtained from 3D hand paths should also be experimented.

The proposed solution was able to achieve better results than some implementations described in the literature (section 2.3.5.4), like the solution proposed by Chen et al. [103] with an accuracy of 85% and the solution proposed by Yoon et al. [104] with an accuracy of 93% in the batch tests and 85% on the online tests. However, we got worst results compared to the solution proposed by Nguyen Dang Binh et al. [105] with an accuracy of 98% and Mahmoud Elmezain et al. [107] with an average accuracy of 98,94% for the isolated gestures and an average accuracy of 95,7% for the continuous gestures.

As a final conclusion it is possible to say that, although the system still needs to evolve and is not a final solution, it was able to successfully integrate posture and gesture recognition with good results during the online tests carried in the laboratory under different types of conditions and different users.

6.5 A Comparative Study of Hand Features for Sign Language Recognition

6.5.1 Experimental Setup

This experiment mail goal was to compare the centroid distance hand features, used in the previous experiment, with a new type of features built from the hand depth image distance values, in order to test which one could achieve better results in Portuguese Sign Language recognition. The distance values, which represent the distance from the object to the camera, can be seen in the image of Figure 100, with different distances represented by different grey values. The distance values could represent, in our opinion, good hand features giving a more possible hand representations with the same viewpoint.

For the centroid distance, in this experiment, only 16 feature values were used. For the distance values, the hand image is resized to be 16x16 pixels in size, giving a final vector with 256 features. So, for this experiment, two types of images were used during the feature extraction phase:

1. the binary hand blob for the centroid distance histogram calculation, as in the previous experiment.
2. the hand distance image for the distance feature vector.

For the centroid distance database, a total of 2170 records with 16 features were used, and for the distance values database, a total of 2488 records with 256 features were used.

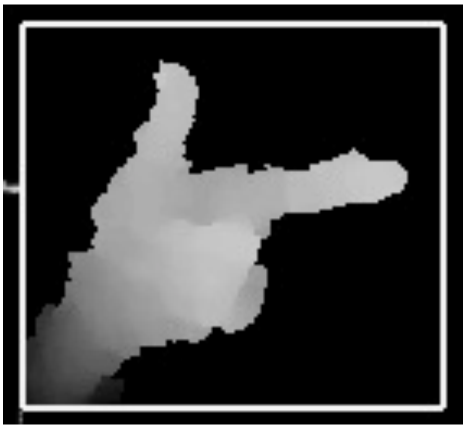
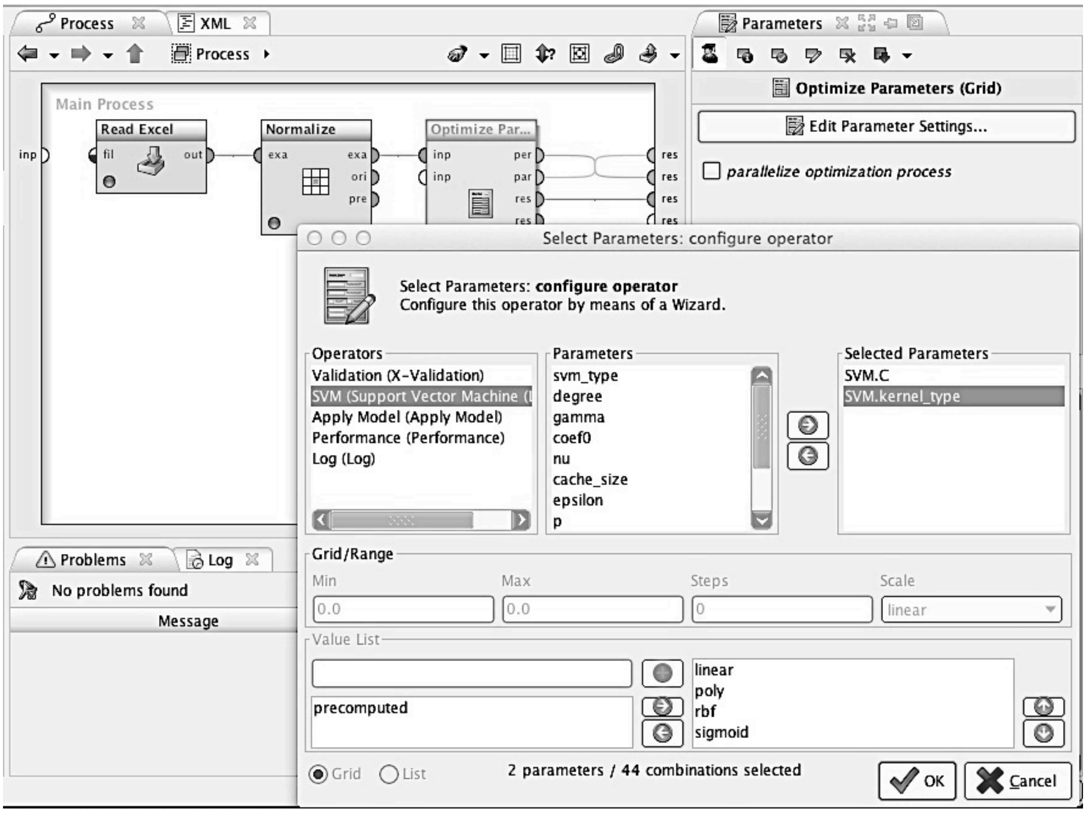


Figure 100. Hand depth image.

For data acquisition and pre-processing, the same application from the previous experiment was used and a function to extract the hand distance values, resize it and build the feature vector was implemented.

All the datasets were converted to Excel files to be imported into RapidMiner for data analysis. The experiment was divided into two phases with the purpose of testing for each dataset which parameters to use. First, the goal was to test in terms of SVM classification, which kernels would achieve the bests results with the two datasets among the following: *linear*, *sigmoid*, *rbf* (radial basis function) and *polynomial*. Although the *rbf* kernel is a reasonable first choice, there are some situations where this is not suitable. In particular, when the number of features is very large, one may just use the linear kernel. This was the reason that led us to perform this test. Second, for each dataset, it was decided to run a parameter optimization test dependent on the chosen kernel type, in order to obtain the best parameters to be implemented in the final solution. Figure 101 shows the RapidMiner process configuration for the kernel and the C (cost value) parameter optimization.

Although in this experiment the datasets contained only features corresponding to the vowels, the model is easily extendable to the rest of the alphabet.



Select Parameters: configure operator

Select Parameters: **configure operator**
Configure this operator by means of a Wizard.

Selected Parameters: SVM.C, SVM.kernel_type

Min: 0.0 Max: 0.0 Steps: 0 Scale: linear

Value List: precomputed | linear, poly, rbf, sigmoid

2 parameters / 44 combinations selected

Figure 101. RapidMiner sign language dataset, kernel type and C value, parameter optimization

6.5.2 Results

The first phase of the experiment, as explained in the previous section, was to learn which kernels behaved better in terms of SVM data classification for the two datasets. As can be seen on Table 28, for both datasets, the obtained best kernel type was the linear kernel, with a difference only on the obtained C parameter. For this reason the second part of the experiment was unnecessary, since the linear kernel does not need any further parameter optimization.

Table 28. Obtained kernel type and C value for the two datasets through parameter optimization

Parameters	Dataset 1	Dataset 2
kernel type	linear	linear
C	1	2

The following table presents the obtained accuracy for each dataset with the corresponding kernel type and C value.

Table 29. Obtained accuracy for each dataset.

Parameters	**Dataset 1**	**Dataset 2**
Kernel type	Linear	linear
C	1	2
accuracy	**99,4%**	**99,6%**

In order to analyse how classification errors are distributed among classes, a confusion matrix was computed for each one of the datasets under study. The resulting confusion matrixes are represented in the following two tables.

Table 30. Centroid distance features confusion matrix

		Actual class				
		1	**2**	**3**	**4**	**5**
Predicted	**1**	**455**	0	0	2	0
	2	0	**394**	1	1	0
	3	0	0	**401**	1	0
	4	4	2	0	**382**	0
	5	0	0	1	0	**439**

Table 31. Distance features confusion matrix

		Actual class				
		1	**2**	**3**	**4**	**5**
Predicted	**1**	**622**	0	8	0	10
	2	0	**543**	1	0	1
	3	0	3	**451**	0	0
	4	0	0	0	**413**	0
	5	0	0	1	0	434

6.5.3 Discussion

The goal of this experiment was to carry a comparative study of two different types of hand features for the problem of sign language recognition using a support vector machine (SVM). It was decided to choose this learning algorithm since it had already shown good results in the previous experiment, in terms of classification, with one of the hand features under study. From Table 29 one can easily see that the two different features gave similar results in terms of classification accuracy, with only a small difference of 0,2%. From the obtained results so far, one can conclude that the new features are not an asset, having the disadvantage of being heavier in terms of number of features and final sizes of the dataset and generated model. The corresponding confusion matrix also shows, that some major classification errors occurred between gesture one, the ‘A’ vowel, and gestures three and five, corresponding to the ‘E’ and ‘U’ vowels respectively.

The centroid distance, on the other hand, being simple in terms of computational complexity gives rise to smaller datasets and a small model file.

Although the results were encouraging, further tests should be carried out, including not only more features from new users as well as the rest of the alphabet in the datasets.

6.6 Summary

This chapter presented a set of experiments carried out during the current thesis, in order to find hand features that could give good results in terms of hand gesture recognition for real-time vision-based systems.

The main goal of the first two and the last experiments was to discover features that could be used in static hand gesture classification. The third experiment also had the intention to test and select features that could be used in dynamic gesture classification.

With the selected features, the intention was to learn models, using machine learning algorithms, that could be used on those systems.

For data analysis, RapidMiner was used, giving us the possibility to rapidly explore different scenarios with different learning algorithms with the created datasets. This tool was a crucial factor in the analysis of all obtained data.

With this set of experiments, it was possible to arrive to a set of features and a learning algorithm that generated a model able to achieve an accuracy of 99.4% in terms of static hand posture classification. It was also possible, with the selected and tested features for dynamic gestures and the Markov models generated for each gesture, to obtain very good classification results, with an average accuracy of 93,72%.

Although the results obtained so far, in terms of static gestures and in terms of dynamic gestures are very encouraging, after analysing them it is possible to conclude that further work may still be carried out in this area. It is however important to point out that the implemented solutions are a solid foundation for the development of generic gesture recognition systems that could be used with any interface for human computer interaction.

7 Conclusions and Future Work

This chapter gives an overview of the work reported in this thesis, along with main results and conclusions. The tools and results achieved are underlined and some development perspectives and future directions are also presented.

7.1 Synthesis of Developed Work

The main goal of this work was the study, project and implement real time gesture recognition solutions, generic enough, with the help of machine learning algorithms, in order to allow their application in a wide range of human-machine interfaces. In this context, the work carried out comprised the:

- Study of the concepts of object classification, feature extraction and the various methodologies proposed by researchers in order to perform gesture classification.
- Analyse of application fields, appropriate for the application of the methodologies for gesture recognition, namely: remote robot control, robotic wheel-chair control, the RoboCup soccer league, the RoboCup@Home league and sign language recognition.
- Study of the approaches of other researchers in the area of the domains of interest, with emphasis on methodologies for static gesture recognition and dynamic gesture recognition.
- Identification of hand features that are simple in terms of computational complexity, to be used in real-time applications.
- Definition of a new and formal language – Wheelchair CommLang – that allows the representation of all the possible gestures that can be used as commands to control a robotic based wheelchair.
- Definition of a new and formal language – Referee CommLang – that allows the representation of all possible gesture combinations (static and dynamic) for the MSL referee commands.
- Development of an application that enables a user to remotely control a robot with a number of simple hand commands.

- Development of an application that enables a user to drive a robotic base wheelchair with a minimum number of finger commands.
- Development of two applications for offline training of static and dynamic hand gestures, in order to create models that can be used for online gesture classification.
- Development of an integrated vision-based hand gesture recognition system, for the classification of the MSL referee commands defined in the new formal language (Referee CommLang).

7.2 Main Results and Conclusions

Hand gestures are a powerful way for human communication, with lots of potential applications in the area of human computer interaction. Vision-based hand gesture recognition techniques have many proven advantages compared with traditional devices. However, hand gesture recognition is a difficult problem and the current work is only a small contribution towards achieving the results needed in the field.

The main objective of this work was to study and implement solutions that could be generic enough, with the help of machine learning algorithms, allowing its application in a wide range of human-computer interfaces, for online gesture recognition. To achieve this, a set of implementations for processing and retrieving hand user information, learn statistical models and able to do online classification were created. The final prototype is a generic solution for a vision-based hand gesture recognition system, that is able to interpret static and dynamic gestures and that can be integrated with any human robot/system interface. The implemented solution, based on supervised learning algorithms, is easily configured to process new hand features or to learn different static and dynamic gestures, while creating statistical models that can be used in any real-time user interface for online gesture classification.

For the problem of static hand posture classification, the hand features that give good classification results were identified, being at the same time simple in terms of computational complexity, for use in any real-time application. The selected features were tested with the help of the RapidMiner tool for machine learning and data mining. That way, it was possible to identify a learning algorithm that was able to achieve very good results in terms of pattern classification, and that was the one used in the final solution.

For the case of dynamic gesture recognition, the choice fell on Hidden Markov Models, due to the nature of the data, gestures, which are time-varying processes. This type of models has proven to be very effective in other areas of application, and had already been applied successfully to the problem of gesture recognition. The evaluation of the trained gestures with this prototype proved that, it was possible to successfully integrate static and dynamic gestures in the generic system for human / computer interaction. Although in the implementation only 2D hand paths were used in order to extract dynamic gesture features, it was shown that for the current system configuration and the set of predefined assumptions, that type of information was enough.

It was also possible to prove through this study, and with the various experiments which were carried, that proper feature selection for image classification is vital for the future performance of the recognition system. It was possible to learn and select sensible features that could be effectively used with machine learning algorithms in order to increase the performance and effectiveness of online static and dynamic gesture classification.

To demonstrate the effectiveness of our vision based gesture recognition system, the proposed methods were evaluated with the Referee CommLang Prototype, able to interpret user commands defined in the new formal language, *the Referee CommLang*, created with the aim of interpreting a set of commands made by a robotic soccer referee. The proposed methods were also evaluated with the Sign Language Recognition prototype, able to interpret the Portuguese Sign Language vowels.

An important aspect to report on the implemented solution has to do with the fact that new users were able to learn the system very quickly and were able to start using it in a normal way after a short period of time, making it a solution that can be easily adapted and applied to other areas of application.

Although to date, the system has been tested in laboratory under various conditions and with various users, it was not possible to test the prototype in a real competition environment with real situations due to calendar restrictions, but it is expected to validate on next year's National Open.

7.3 Limitations

Despite all the work carried out, the final solution has some limitations mainly due to the type of camera used. Although the camera has advantages in some aspects that were explored in the development of the project, it has however some limitations with implications in the final work. Following are present the main system limitations.

- The system is only prepared to work immediately with the camera used in the experiments.
- Hand pose must be defined with a bare hand and not occluded by other objects, i.e., the selected hand features are not robust to partial occlusions.
- The system is not capable of viewpoint independent hand gesture recognition.
- The system needs that the user is positioned in front of the camera, within a predefined perimeter area and within a defined distance range due to camera limitations.
- The system does not allow continuous dynamic gestures.
- The current system does allow the recognition of static gestures during the execution of dynamic gesture, due to camera limitations.
- The system only works in indoor environments due to the characteristics of the camera used.

7.4 Major Contributions

The major contributions of this thesis are:

- Definition of a new and formal command language which may be used in any type of vision-based system, able to interpret a large number of commands. Those commands can be used to control, among others, a robotic wheelchair or a robotic soccer game. This language was defined using Bakus Naur form with the advantage of being simple in its syntax and easily adapted to other areas of application.

- Project and implementation of a system that is able to remotely drive a robot or control a electronic device using a finite set of hand gestures. The system is flexible enough to be easily adapted, with minor changes, to control distinct robots performing remote operations and other kinds of systems.
- Implementation of two applications able to train and learn statistical based models, for static or dynamic gesture recognition. The learned models may then be integrated in any vision-based system for real time hand gesture classification.
- Implementation of a generic system able to recognize static, dynamic or a combination of both types of gestures to classify any sequence of gestures in real-time, using previously learned models. The classification is controlled by the command definition with the new formal language.
- Definition of methodologies enabling the integration of machine learning algorithms to increase the performance and effectiveness of real time static and dynamic gesture classification systems.
- Development of three fully-functional gesture based control applications for robot remote control, MSL robotic soccer refereeing and sign language recognition, in order to validate the approach.

7.5 Development Perspectives and Future Work

Although the objectives of this thesis are fulfilled, many situations arose during the study that should be implemented and some others experimented and explored.

So, this section identifies some possible developments and further work that on one hand can be implemented as a complement to what was developed during this thesis, or as promising and worth exploring areas.

In short, as major development prospects and further work it is suggested:

- To improve the data acquisition phase, thus giving the possibility to improve the type of hand features extracted, being able to construct more reliable statistical models for online classification. In order to pursue this objective, ways must be found to reduce the noise present in the type of cameras used in the present study, or possibly try new cameras available at the moment, with improved depth resolution and with a higher frame frequency.

- Build an add-on that would allow to easily configure system settings in terms of number and type of static and dynamic gestures.
- Implement the *Command Definition Language* module. This module would allow to easily managing settings related to the definition of the new commands.
- Explore other machine learning algorithms applied to the problem of hand gesture classification and compare obtained results.
- Include not only the possibility of 3D gestures but also to work with several cameras to thereby obtain a full 3D environment and achieve view-independent recognition, thus eliminating some limitations of the current system.
- Explore the possibility of applying stereo vision instead of only depth range cameras, applied to human / computer interaction and particularly to hand gesture recognition.
- Introduce gesture recognition with both hands, enabling the creation of more natural interaction environments.
- Investigate and try to find more reliable solutions for the identification of the beginning and end of a gesture.
- Build systems that are able to recognize continuous gestures, i.e., without the need to introduce pauses for gesture or command construction.
- Explore reinforcement learning as a way to start with a reduced number of hand features per gesture, reducing the time to learn the models, and be able to learn with user interaction, possibly using multimodal dialog strategies.
- Explore unsupervised learning applied to gesture recognition. Give the robot/system the possibility to learn by interaction with the user, again with the possibility of multimodal strategies.

As a final conclusion one can say that although there is still much to do in the area, the implemented solutions are a solid foundation for the development of generic gesture recognition systems that could be used with any interface for human computer interaction. The interface language can be redefined and the system can be easily configured to train different set of gestures that can be easily integrated with any desired solution.

References

[1] M. T. Committe. (2013, April 2013). Middle Size Robot League Rules and Regulations for 2013. Available: http://wiki.robocup.org/wiki/Middle_Size_League - Rules

[2] L. P. Reis, F. Almeida, L. Mota, and N. Lau, "Coordination in Multi-robot Systems: Applications in Robotic Soccer," in Agents and Artificial Intelligence. vol. 358, J. Filipe and A. Fred, Eds., ed: Springer Berlin Heidelberg, 2013, pp. 3-21.

[3] RoboCup. (2013, April 2013). RoboCup@Home. Available: http://www.robocup.org/robocup-home/

[4] J. R. Chowdhury, "Kinect Sensor for Xbox Gaming," M.Tech CSE, IIT Kharagpur, 2012.

[5] P. Trigueiros and F. Ribeiro, "Vision-based Hand WheelChair Control," in 12th International Conference on Autonomous Robot Systems and Competitions, Guimarães, Portugal, 2012, pp. 39-43.

[6] P. Trigueiros, F. Ribeiro, and G. Lopes, "Vision-based hand segmentation techniques for human-robot interaction for real-time applications," in III ECCOMAS Thematic Conference on Computational Vision and Medical Image Processing, Olhão, Algarve, Portugal, 2011, pp. 31-35.

[7] P. Trigueiros, F. Ribeiro, and L. P. Reis, "Vision-based Gesture Recognition System for Human-Computer Interaction," in IV ECCOMAS Thematic Conference on Computational Vision and Medical Image Processing, Funchal. Madeira, 2013.

[8] P. Trigueiros, F. Ribeiro, and L. P. Reis, "A Comparative Study of different image features for hand gesture machine learning," in 5th International Conference on Agents and Artificial Intelligence, Barcelona, Spain, 2013.

[9] P. Trigueiros, F. Ribeiro, and L. P. Reis, "Vision Based Referee Sign Language Recognition System for the RoboCup MSL League," in 17th annual RoboCup International Symposium, Eindhoven, Holland, 2013.

[10] G. R. S. Murthy and R. S. Jadon, "A Review of Vision Based Hand Gestures Recognition," International Journal of Information Technology and Knowledge Management, vol. 2, pp. 405-410, July-December 2009 2009.

[11] S. Ong and S.Ranganath, "Automatic sign language analysis: A survey and the future beyond lexical meaning," IEEE Trans. Pattern Analysis ans Machine Intelligence, vol. 27, pp. 873-891, June 2005 2005.

[12] S. Conseil, S. Bourenname, and L. Martin, "Comparison of Fourier Descriptors and Hu Moments for Hand Posture Recognition," presented at the 15th European Signal Processing Conference (EUSIPCO), Poznan, Poland, 2007.

[13] S. Mitra and T. Acharya. (2007, May 2007) Gesture recognition: A Survey. IEEE Transactions on Systems, Man and Cybernetics. 311-324. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4154947

[14] G. Simion, V. Gui, and M. Otesteanu, "Vision Based Hand Gesture Recognition: A Review," International Journal of Circuits, Systems and Signal Processing, vol. 6, pp. 275-282, 2012.

[15] S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," Artificial Intelligence Review, pp. 1-54, 2012/11/01 2012.

[16] R. Z. Khan and N. A. Ibraheem, "Hand Gesture Recognition: A Literature Review," International Journal of Artificial Intelligence & Applications, vol. 3, pp. 161-174, July 2012 2012.

[17] Y. Mingqiang, K. Idiyo, and R. Joseph, "A Survey of Shape Feature Extraction Techniques," Pattern Recognition, pp. 43-90, November 2008 2008.

[18] S. Bourennane and C. Fossati, "Comparison of shape descriptors for hand posture recognition in video," Signal, Image and Video Processing, vol. 6, pp. 147-157, 2010.

[19] R. Y. Tara, P. I. Santosa, and T. B. Adji, "Sign Language Recognition in Robot Teleoperation using Centroid Distance Fourier Descriptors," International Journal of Computer Applications, vol. 48, June 2012 2012.

[20] J. A. Zondag, T. Gritti, and V. Jeanne, "Practical study on real-time hand detection," in 3rd International Conference on Affective Computing and Intelligent Interaction and Workshops Amsterdam, the Netherlands, 2009, pp. 1-8.

[21] Z. Yafei, W. Weidong, and W. Yuehai, "A real-time hand gesture recognition method," in International Conference on Electronics, Communications and Control, Ningbo, China, 2011, pp. 2475-2478.

[22] M. Xingbao, L. Jing, and D. Yingchun, "An extended HOG model: SCHOG for human hand detection," in International Conference on Systems and Informatics, Yantai, China, 2012, pp. 2593-2596.

[23] M. PietiKainen, A. Hadid, G. Zhao, and T. Ahonen, Computer Vision Using Local Binary Patterns. London: Springer-Verlag, 2011.

[24] D. Zhang and G. Lu, "A Comparative Study on Shape Retrieval Using Fourier Descriptors with Different Shape Signatures," Journal of Visual Communication and Image Representation, pp. 41-60, // 2003.

[25] D. Zhang and G. Lu, "A comparative Study of Fourier Descriptors for Shape Representation and Retrieval," in Proc. of 5th Asian Conference on Computer Vision (ACCV), Melbourne, Australia, 2002, pp. 646--651.

[26] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," Foundations and Trends in Computer Graphics and Vision, vol. 3, pp. 177-280, 2007.

[27] C. Harris and M. Stephens, "A combined corner and edge detector," in The Fourth Alvey Vision Conference, 1988, pp. 147–151.

[28] G. Bradski and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, 1st ed.: O'Reilly Media, 2008.

[29] J. Shi and C. Tomasi, "Good Features to Track," presented at the Internacional Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 1994.

[30] M. Roth, K. Tanaka, C. Weissman, and W. Yerazumis. (1998, May-1998) Computer Vision for Interactive Computer Graphics. IEEE Computer Graphics And Applications. 42-53.

[31] W. T. Freeman and M. Roth, "Orientation Histograms for Hand Gesture Recognition," Mitsubishi Electric Research Laboratories, Cambridge Research CenterDecember 1994 1994.

[32] R. C. Gonzalez and R. E. Woods, Digital Image Processing: Addison-Wesley Longman Publishing, 2001.

[33] W. E. Snyder and H. Qi, Machine Vision: Cambridge University Press, 2004.

[34] E. R. Davies, Machine Vision - Theory, Algorithms, Practicalities, 3rd Edition ed.: Morgan Kaufmann, 2005.

[35] T. Ojala, M. PeitiKainen, and T. Maenpã, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," IEEE Trans. Pattern Analysis ans Machine Intelligence, vol. 24, pp. 971-987, July 2002 2002.

[36] M. Hruz, J. Trojanova, and M. Zelezny, "Local binary pattern based features for sign language recognition," Pattern Recognition and Image Analysis, vol. 21, pp. 398-401, 2011.

[37] D. Unay, A. Ekin, M. Cetin, R. Jasinschi, and A. Ercil, "Robustness of Local Binary Patterns in Brain MR Image Analysis," in 29th Annual Conference of the IEEE EMBS, Lyon, France, 2007, pp. 2098-2101.

[38] M. Pietikainen, T. Ojala, and Z. Xu, "Rotation-Invariant Texture Classification using Feature Distributions," Pattern Recognition, vol. 33, pp. 43-52, 2000.

[39] M. Treiber, An Introduction to Object Recognition: Springer, 2010.

[40] F. Y. Shih, Image Processing and Pattern Recognition: Fundamentals and Techniques. Canada: Wiley and Sons, 2008.

[41] M. Frigo, "A Fast Fourier Transform Compiler," in ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '99), Atlanta, Georgia, 1999.

[42] Z. Hanning, D. J. Lin, and T. S. Huang, "Static Hand Gesture Recognition based on Local Orientation Histogram Feature Distribution Model," in Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on, 2004, pp. 161-161.

[43] S. Liang, W. Guijin, Y. Anbang, L. Xinggang, and C. Xiujuan, "Hand posture recognition in video using multiple cues," in IEEE International Conference on Multimedia and Expo, New York, NY, USA, 2009, pp. 886-889.

[44] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in International Conference on Computer Vision & Pattern Recognition, Grenoble, France, 2005, pp. 886-893.

[45] M. Isard and A. Blake, "Condensation - Conditional Density Propagation for Visual Tracking," International Journal of Computer Vision, vol. 29, pp. 5-28, 1998/08/01 1998.

[46] J. Friedman, T. Hastie, and R. Tibshirani, "Additive Logistic Regression: a Statistical View of Boosting," The Annals of Statistics, vol. 38, pp. 337-374, 2000.

[47] Y. Freund and R. Schapire, "Experiments with a New Boosting Algorithm," in 13th International Conference on Machine Learning, Bari, Italy, 1996, pp. 148-156.

[48] M. B. Kaaniche and F. Bremond, "Tracking HoG Descriptors for Gesture Recognition," in 6th IEEE International Conference on Advanced Video and Signal Based Surveillance, Genova, Italy, 2009, pp. 140-145.

[49] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local SVM approach," in 17th International Conference on Pattern Recognition, Cambridge, UK, 2004, pp. 32-36.

[50] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," presented at the Proceedings of the 9th European conference on Computer Vision - Volume Part I, Graz, Austria, 2006.

[51] M. Hruz, J. Trojanova, and M. Zelezny, "Local binary pattern based features for sign language recognition," Pattern Recognit. Image Anal., vol. 21, pp. 398-401, 2011.

[52] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," Pattern Recognition, vol. 29, pp. 51-59, 1// 1996.

[53] J. Gao and Q. Cao, "Adaptive HOG-LBP Based Learning for Palm Tracking," in 2nd International Conference on Computer and Information Applications, Taiyuan, Shanxi, China, 2012, pp. 543-546.

[54] A. L. C. Barczak, A. Gilman, H. H. Reyes, and T. Susnjak, "Analysis of Feature Invariance and Discrimination for Hand Images: Fourier Descriptors versus Moment Invariants," presented at the International Conference Image and Vision Computing, New Zeland, 2011.

[55] M.-K. Hu, "Visual pattern recognition by moment invariants," Information Theory, IRE Transactions on, vol. 8, pp. 179-187, february 1962.

[56] J. Triesch and C. v. d. Malsburg, "Robust Classification of Hand Postures against Complex Backgrounds," in International Conference on Automatic Face and Gesture Recognition, Killington, Vermont, USA, 1996, pp. 170-175.

[57] D. G. Lowe, "Object Recognition from local scale-invariant features," in International Conference on Computer Viison, Corfu, Greece, 1999, pp. 1150-1157.

[58] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-Up Robust Features (SURF)," Computer Vision and Image Understanding, vol. 110, pp. 346-359, June 2008 2008.

[59] C.-C. Wang and K.-C. Wang, "Hand Posture Recognition Using Adaboost with SIFT for Human Robot Interaction," in Proceedings of the International Conference on Advanced Robotics Jeju, Korea, 2008.

[60] D. Q. Huynh, "Evaluation of Three Local Descriptors on Low Resolution Images for Robot Navigation," in 24th International Conference Image and Vision Computing, Wellington, New Zealand, 2009, pp. 113-118

[61] T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Data Mining Researchers," ed, 2003.

[62] M. Kolsch and M. Turk, "Robust hand detection," in 6th IEEE International Conference on Automatic Face and Gesture Recognition, Seoul, South Korea, 2004, pp. 614-619.

[63] P. Viola and M. Jones, "Robust Real-time Object Detection," International Journal of Computer Vision - to appear, // 2002.

[64] I. Millington and J. Funge, Artificial Intelligence for Games, second edition ed.: Elsevier, 2009.

[65] F. Camastra and A. Vinciarelli, Machine Learning for Audio, Image and Video Analysis: Springer, 2008.

[66] E. Alpaydin, Introduction to Machine Learning: MIT Press, 2004.

[67] R. Herbrich, Learning Kernel Classifiers: Theory and Algorithms: MIT Press, 2003.

[68] B. M. Faria, N. Lau, and L. P. Reis, "Classification of Facial Expressions Using Data Mining and machine Learning Algorithms," in 4ª Conferência Ibérica de Sistemas e Tecnologias de Informação, Póvoa de Varim, Portugal, 2009, pp. 197-206.

[69] N. E. Gillian, "Gesture Recognition for Musician Computer Interaction," Doctor of Philosophy, Music Department, Faculty of Arts, Humanities and Social Sciences, Belfast, 2011.

[70] B. M. Faria, L. P. Reis, N. Lau, and G. Castillo, "Machine Learning Algorithms applied to the Classification of Robotic Soccer Formations ans Opponent Teams," presented at the IEEE Conference on Cybernetics and Intelligent Systems (CIS), Singapore, 2010.

[71] A. Mannini and A. M. Sabatini, "Machine learning methods for classifying human physical activity from on-body accelerometers," Sensors, vol. 10, pp. 1154-1175, 2010.

[72] R. Vicen-Bueno, R. Gil-Pita, M. P. Jarabo-Amores, and F. López-Ferreras, "Complexity Reduction in Neural Networks Appplied to Traffic Sign Recognition Tasks," 2004.

[73] S. Maldonado-Báscon, S. Lafuente-Arroyo, P. Gil-Jiménez, and H. Gómez-Moreno. (2007, June 2007) Road-Sign detection and Recognition Based on Support Vector Machines. IEEE Transactions on Intelligent Transportation Systems. 264-278.

[74] P. Trigueiros, F. Ribeiro, and L. P. Reis, "A comparison of machine learning algorithms applied to hand gesture recognition," in 7th Iberian Conference on Information Systems and Technologies, Madrid, Spain, 2012, pp. 41-46.

[75] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE, vol. 77, pp. 257-286, 1989.

[76] A. K. Jain, R. P. W. Duin, and J. Mao, "Statisitical Pattern Recognition: A Review," IEEE Trans. Pattern Analysis ans Machine Intelligence, vol. 22, pp. 4-36, Janurary 2000 2000.

[77] I. H. Witten, E. Frank, and M. A. Hall, Data Mining - Pratical Machine Learning Tools and Techniques, Third Edition ed.: Elsevier, 2011.

[78] T. Ivry and S. Michal. (8 Nov 2012). License Plate Number Recognition Using Artificial Neural Network. Available: http://www.cs.bgu.ac.il/~icbv061/StudentProjects/ICBV061/ICBV-2006-1-TorIvry-ShaharMichal/index.php

[79] WikiBooks. (2012, 8 Nov 2012). Artificial Neural Networks/Activation Functions. Available: http://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions

[80] S. Haykin, Neural Networks - A Comprehensive Foundation, 2nd ed. Delhi: Prentice Hall, 1999.

[81] F. Gorunescu, Data Mining - Concepts, Models and Techniques vol. 12: Springer-Verlag, 2011.

[82] A. Ben-Hur and J. Weston, "A User's Guide to Support Vector Machines," in Data Mining Techniques for the Life Sciences. vol. 609, ed: Humana Press, 2008, pp. 223-239.

[83] S. Theodoridis and K. Koutroumbas, Pattern Recognition, 4th Edition: Elsevier, 2009.

[84] D. S. Sayad. (2010, 8 Nov 2012). Support Vector Machine - Classification (SVM). Available: http://www.saedsayad.com/support_vector_machine.htm

[85] G. A. Fink, Markov Models for Pattern recognition - From Theory to Applications: Springer, 2008.

[86] Y. Jufeng, X. Jing, L. Mingda, Z. Dingzhen, and W. Congchao, "A real-time command system based on hand gesture recognition," in 7th International Conference on Natural Computation, Shanghai, China 2011, pp. 1588-1592.

[87] H. Lahamy and D. D. Lichti, "Performance analysis of different classification methods for hand gesture recognition using range cameras," in Videometrics, Range Imaging, and Applications XI, Munich, Germany, 2011.

[88] T. H. H. Maung, "Real-Time Hand Tracking and Gesture Recognition System Using Neural Networks," Proceedings of World Academy of Science: Engineering & Technology, vol. 50, pp. 466-470, 2009.

[89] T. Ahmed, "A Neural Network based Real Time Hand Gesture Recognition System," International Journal of Computer Applications, vol. 59, pp. 17-22, December 2012 2012.

[90] P. Mekala, J. Fan, W.-C. Lai, and C.-W. Hsue, "Gesture Recognition Using Neural Networks Based on HW/SW Cosimulation Platform," Advances in Software Engineering, vol. 2013, p. 13, 2013.

[91] H. Hasan and S. Abdul-Kareem, "Static hand gesture recognition using neural networks," Artificial Intelligence Review, pp. 1-35, 2012/01/01 2012.

[92] Y. S. Abu-Mostafa and D. Psaltis, "RecognitiveAspectsofMoment Invariants," IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, vol. 6, pp. 698-706, 1984.

[93] H. Ching-Tang, Y. Cheng-Hsiang, H. Kuo-Ming, C. Li-Ming, and K. Chin-Yen, "A real time hand gesture recognition system based on DFT and SVM," in 8th International Conference on Information Science and Digital Content Technology, Jeju, Korea, 2012, pp. 490-494.

[94] G. Bradski, "Computer Vision Face Tracking For Use in a Perceptual User Interface," Intel Technology Journal, 1998.

[95] Y. H. Liu, "Feature Analysis and Classifier Design and Their Applications to Pattern Recognition and Data Mining," Ph.D., National Taiwan University, R.O.C., 2003.

[96] C. Yen-Ting and T. Kuo-Tsung, "Multiple-angle Hand Gesture Recognition by Fusing SVM Classifiers," in IEEE International Conference on Automation Science and Engineering, Washington DC, USA, 2007, pp. 527-530.

[97] Y. Liu and P. Zhang, "An Automatic Hand Gesture Recognition System Based on Viola-Jones Method and SVMs," in Computer Science and Engineering, 2009. WCSE '09. Second International Workshop on, 2009, pp. 72-76.

[98] J.-H. Ahn, C. Choi, and T.-S. Kim, "Gesture Recognition using Singular State Analysis and the Support Vector Machine," Pacific Science Review, vol. 9, pp. 29-34, 2007.

[99] N. H. Dardas and N. D. Georganas, "Real-Time Hand Gesture Detection and Recognition Using Bag-of-Features and Support Vector Machine Techniques," Instrumentation and Measurement, IEEE Transactions on, vol. 60, pp. 3592-3607, 2011.

[100] Y.-G. Jiang, C.-W. Ngo, and J. Yang, "Towards optimal bag-of-features for object categorization and semantic video retrieval," presented at the 6th International Conference on Image and Video Retrieval, Amsterdam, The Netherlands, 2007.

[101] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories," presented at the Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2, 2006.

[102] C.-C. Hsieh and D.-H. Liou, "Novel Haar features for real-time hand gesture recognition using SVM," Journal of Real-Time Image Processing, pp. 1-14, 2012/11/01 2012.

[103] 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, pp. 745-758, 2003.

[104] H.-S. Yoon, J. Soh, Y. J. Bae, and H. Seung Yang, "Hand gesture recognition using combined features of location, angle and velocity," Pattern Recognition, vol. 34, pp. 1491-1501, // 2001.

[105] N. D. Binh, E. Shuichi, and T. Ejima, "Real-Time Hand Tracking and Gesture Recognition System," in Proceedings of International Conference on Graphics, Vision and Image Cairo - Egypt, 2005, pp. 362--368.

[106] O. E. Agazzi and S.-s. Kuo, "Pseudo Two-Dimensional Hidden Markov Models for Document Recognition," AT&T Technical Journal, vol. 72, pp. 60-72, 1993.

[107] M. Elmezain, A. Al-Hamadi, J. Appenrodt, and B. Michaelis, "A Hidden Markov Model-based Continuous Gesture Recognition System for Hand Motion Trajectory," in 19th International Conference on Pattern Recognition, Tampa, Florida, USA, 2008, pp. 1-4.

[108] O. Rashid, A. Al-Hamadi, and B. Michaelis, "A framework for the integration of gesture and posture recognition using HMM and SVM," in IEEE International Conference on Intelligent Computing and Intelligent Systems, Shangai, China, 2009, pp. 572-577.

[109] S. Bilal, R. Akmeliawati, A. Shafie, and M. Salami, "Hidden Markov model for human to computer interaction: a study on human hand gesture recognition," Artificial Intelligence Review, pp. 1-22, 2011/12/01 2011.

[110] Anand.H.Kulkarni and Sachin.A.Urabinahatti, "Performance Comparison of Three Different Classifiers for HCI Using hand Gestures," International Journal of Modern Engineering Research (IJMER), vol. 2, pp. 2857 - 2861, uly-Aug 2012 2012.

[111] A. Khotanzad and H. Yaw Hua, "Invariant image recognition by Zernike moments," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 12, pp. 489-497, 1990.

[112] H. H. Avilés, W. Aguilar, and L. A. Pineda, "On the Selection of a Classification Technique for the Representation and Recognition of Dynamic Gestures.," in Advances in Artificial Intelligence - IBERAMIA 2008, 11th Ibero-American Conference on AI, Lisboa, Portugal, 2008, pp. 412-421.

[113] M. Oshita and T. Matsunaga, "Automatic learning of gesture recognition model using SOM and SVM," in International Conference on Advances in Visual Computing, Las Vegas, NV, USA, 2010, pp. 751-759.

[114] T. Kohonen, Self-Organizing Maps vol. 30: Springer-Verlag New York, Inc., 2001.

[115] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, pp. 273-297, 1995/09/01 1995.

[116] G. Bailador, D. Roggen, G. Tröster, and G. Triviño, "Real time gesture recognition using continuous time recurrent neural networks," in 2nd International Conference on Body Area Networks, Florence, Italy, 2007, pp. 1-8.

[117] J. C. Gallagher and J. M. Fiore, "Continuous time recurrent neural networks: a paradigm for evolvable analog controller circuits," in National Aerospace and Electronics Conference, Dayton, Ohio, USA, 2000, pp. 299-304.

[118] R. D. Beer, "On the dynamics of small continuous-time recurrent neural networks," Adapt. Behav., vol. 3, pp. 469-509, 1995.

[119] A. Chaudhary, J. L. Raheja, K. Das, and S. Raheja, "Intelligent Approaches to interact with Machines using Hand Gesture Recognition in Natural way: A Survey," International Journal of Computer Science & Engineering Survey, vol. 2, pp. 122-133, Feb 2011 2011.

[120] M. Hasanuzzaman, V. Ampornaramveth, T. Zhang, M. A. Bhuiyan, Y. Shirai, and H.Ueno, "Real-time Vision-based Gesture Recognition for Human Robot Interaction," in IEEE International Conference on Robotics and Biomimetics, Shenyang, China, 2004, pp. 413-418.

[121] T. Huang and V. Pavlovic, "Hand Gesture Modeling, Analysis, and Synthesis," in In Proc. of IEEE International Workshop on Automatic Face and Gesture Recognition, 1995, pp. 73-79.

[122] J.-H. Yoon, J.-S. Park, and M. Y. Sung, "Vision-Based bare-hand gesture interface for interactive augmented reality applications," in 5th international conference on Entertainment Computing, Cambridge, UK, 2006, pp. 386-389.

[123] V. Buchmann, S. Violich, M. Billinghurst, and A. Cockburn, "FingARtips: gesture based direct manipulation in Augmented Reality," presented at the 2nd international Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, Singapore, 2004.

[124] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock, "Gestures as point clouds: a $P recognizer for user interface prototypes," presented at the 14th ACM international conference on Multimodal interaction, Santa Monica, California, USA, 2012.

[125] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes," presented at the Proceedings of the 20th annual

ACM symposium on User interface software and technology, Newport, Rhode Island, USA, 2007.

[126] Y. Li, "Protractor: a fast and accurate gesture recognizer.," presented at the Conference on Human Factors in Computing Systems, Atlanta, Georgia, USA, 2010.

[127] S. Kratz and M. Rohs, "Protractor3D: a closed-form solution to rotation-invariant 3D gestures," presented at the 16th International Conference on Intelligent User Interfaces, Palo Alto, CA, USA, 2011.

[128] T. Kim. (2008, 29-03-2013). In-Depth: Eye To Eye - The History Of EyeToy. Available: http://www.gamasutra.com/php-bin/news_index.php?story=20975

[129] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton, and P. Presti, "American sign language recognition with the kinect," presented at the 13th International Conference on Multimodal Interfaces, Alicante, Spain, 2011.

[130] G. A. t. Holt, M. J. T. Reinders, E. A. Hendriks, H. d. Ridder, and A. J. v. Doorn, "Influence of handshape information on automatic sign language recognition," in 8th International Conference on Gesture in Embodied Communication and Human-Computer Interaction, Bielefeld, Germany, 2010, pp. 301-312.

[131] OpenNI. (2013). The standard framework for 3D sensing. Available: http://www.openni.org/

[132] B. Glennoah, "Microsoft Kinect Sensor Evaluation," NASA 20110022972, August 05 2011.

[133] L. PrimeSense. (2013, September 2012). PrimeSense.

[134] B. Widenhofer. (2010). Inside Xbox 360’s Kinect controller. Available: http://www.eetimes.com/document.asp?doc_id=1281322

[135] L. Hacker. (2012, 8 Nov 2012). LabView Kinect Interface. Available: http://labviewhacker.com/doku.php?id=projects:lv_kinect_interface:lv_kinect_interface

[136] M. R. Andersen, T. Jensen, P. Lisouski, A. K. Mortensen, M.K. Hansen, T. Gregersen, et al., "Kinect Depth Sensor Evaluation for Computer Vision Applications," Department of Engineering – Electrical and Computer Engineering, Aarhus UniversityFebruary 2012 2012.

[137] M. Camplani and L. Salgado, "Efficient spatio-temporal hole filling strategy for Kinect depth maps," Three-Dimensional Image Processing (3DIP) and Applications II, vol. 8290, p. 10, 2012.

[138] B. Bongalon. (2011). Experiment to remove noise in Kinect depth maps. Available: http://borglabs.com/blog/experiment-to-remove-noise-in-kinect-depth-maps

[139] P. Garg, N. Aggarwal, and S. Sofat, "Vision Based Hand Gesture Recognition," World Academy of Science, Engineering and Techonology, pp. 972-977, 2009.

[140] J. J. Stephan and S. Khudayer, "Gesture Recognition for Human-Computer Interaction (HCI)," International Journal of Advancements in Computing Technology, vol. 2, pp. 30-35, 2010.

[141] J. P. Wachs, H. Stern, and Y. Edan, "Cluster labeling and parameter estimation for the automated setup of a hand-gesture recognition system," Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 35, pp. 932-944, 2005.

[142] H. Kauppinen, T. Seppanen, and M. Pietikainen, "An experimental comparison of autoregressive and Fourier-based descriptors in 2D shape classification," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 17, pp. 201-207, 1995.

[143] D. C. Montgomery and G. C. Runger, Applied Statistics and Probability for Engineers: Wiley, 1994.

[144] S. Theodoridis and K. Koutroumbas, An Introduction to Pattern Recognition: A Matlab Approach: Academic Press, 2010.

[145] D. E. King, "Dlib-ml: A Machine Learning Toolkit," Journal of Machine Learning Research, vol. 10, pp. 1755-1758, 2009.

[146] L. R. Rabiner and B. H. Juang, "An introduction to hidden Markov models," IEEE ASSp Magazine, 1986.

[147] Y. Wu and T. S. Huang, "Vision-Based Gesture Recognition: A Review," presented at the Proceedings of the International Gesture Workshop on Gesture-Based Communication in Human-Computer Interaction, 1999.

[148] L. Rung-Huei and O. Ming, "A real-time continuous gesture recognition system for sign language," in Third IEEE International Conference on Automatic Face and Gesture Recognition, Nara, Japan, 1998, pp. 558-567.

[149] M. Buckland, Programming Game AI by Example: Wordware Publishing, Inc., 2005.

[150] A. F. Ribeiro, G. Lopes, J. Costa, J. P. Rodrigues, B. Pereira, J. Silva, et al., "Minho MSL : a new generation of soccer robots," presented at the Robocup 2011, Istambul, 2011.

[151] F. Ribeiro. (2006, April, 2013). Grupo de Automação e Robótica. Available: http://www.robotica.dei.uminho.pt

[152] A. F. Ribeiro. (2007) ENIGMA : Cadeira de Rodas Omnidireccional. Robótica. 50-51.

[153] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," SIGGRAPH 2001, 2001.

[154] P. S. Maybeck, Stochastic models, estimation, and control vol. 1: Academic Press, 1979.

[155] S. Malik, "Real-time Hand tracking and Finger Tracking for Interaction," 2003.

[156] L. P. Reis and N. Lau, "COACH UNILANG - A Standard Language for Coaching a (Robo) Soccer Team," in RoboCup 2001: Robot Soccer World Cup V. vol. 2377, A. Birk, S. Coradeschi, and S. Tadokoro, Eds., ed: Springer Berlin Heidelberg, 2002, pp. 183-192.

[157] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, et al. (1960, January 1963) Revised Report on the Algorithmic Language ALGOL 60. Communications of the ACM. 1-17. Available: http://doi.acm.org/10.1145/366193.366201

[158] F. Mónica, S. Vasconcelos, L. P. Reis, and N. Lau, "Evaluation of distinct input methods of an intelligent wheelchair in simulated and real environments: a performance and usability study," 20130808 DCOM- 20130827 2013.

[159] T. Niemueller. (2013). RoboCup Logistics League - Referee Box. Available: http://www.robocup-logistics.org/refbox

[160] K. Murphy. (1998). Hidden Markov Model (HMM) Toolbox for Matlab. Available: Hidden Markov Model (HMM) Toolbox for Matlab

[161] R. Miner. (December 2011). RapidMiner : Report the Future. Available: http://rapid-i.com/content/view/181/196/

[162] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," ACM Transactions on Intelligent Systems and Technology, vol. 2, p. 27, 2011.

[163] C.-W. Hsu and C.-J. Lin, "A Comparison of Methods for Multiclass Support Vector Machines," IEEE Transactions on Neural Networks, vol. 13, pp. 415-425, March, 2002 2002.

[164] S. Knerr, L. Personnaz, and G. Dreyfus, "Single-layer learning revisited: a stepwise procedure for building and training a neural network," in Neurocomputing. vol. 68, F. Soulié and J. Hérault, Eds., ed: Springer Berlin Heidelberg, 1990, pp. 41-50.

[165] S. V. Stehman, "Selecting and interpreting measures of thematic classification accuracy," Remote Sensing of Environment, vol. 62, pp. 77-89, 10// 1997.

[166] K. Murphy. (1998). Hidden Markov Model (HMM) Toolbox for Matlab. Available: http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html