REAL-TIME HAND GESTURE RECOGNITION EXPLOITING MULTIPLE 2D AND 3D CUES

ADVISOR: Ch.mo Prof. Guido Maria Cortelazzo
PH.D. CANDIDATE: MSc. Fabio Dominio

Ph.D. School in Information and Communication Science and Technologies

Academic Year: 2013-2014

UNIVERSITY OF PADUA
Department of Information Engineering
Ph.D. School in Information and Communication
Science and Technologies

PH.D THESIS

REAL-TIME HAND GESTURE RECOGNITION EXPLOITING MULTIPLE 2D AND 3D CUES

ADVISOR: Prof. Guido Maria Cortelazzo

PH.D. CANDIDATE: MSc. Fabio Dominio

Academic Year: 2013-2014

Abstract

The recent introduction of several 3D applications and stereoscopic display technologies has created the necessity of novel human-machine interfaces. The traditional input devices, such as keyboard and mouse, are not able to fully exploit the potential of these interfaces and do not offer a natural interaction. Hand gestures provide, instead, a more natural and sometimes safer way of interacting with computers and other machines without touching them. The use cases for gesture-based interfaces range from gaming to automatic sign language interpretation, health care, robotics, and vehicle automation.

Automatic gesture recognition is a challenging problem that has been attaining a growing interest in the research field for several years due to its applications in natural interfaces. The first approaches, based on the recognition from 2D color pictures or video only, suffered of the typical problems characterizing such type of data. Inter occlusions, different skin colors among users even of the same ethnic group and unstable illumination conditions, in facts, often made this problem intractable. Other approaches, instead, solved the previous problems by making the user wear sensorized gloves or hold proper tools designed to help the hand localization in the scene.

The recent introduction in the mass market of novel low-cost range cameras, like the Microsoft Kinect™, Asus XTION, Creative Senz3D, and the Leap Motion, has opened the way to innovative gesture recognition approaches exploiting the geometry of the framed scene. Most methods share a common gesture recognition pipeline based on firstly identifying the hand in the framed scene, then extracting some relevant features on the hand samples and finally exploiting suitable machine learning techniques in order to recognize the performed gesture from a predefined "gesture dictionary".

This thesis, based on the previous rationale, proposes a novel gesture recognition framework exploiting both color and geometric cues from low-cost color and range cameras. The dissertation starts by introducing the automatic hand gesture recognition problem, giving an overview of the state-of-art algorithms and the recognition pipeline employed in this work. Then, it briefly describes the major low-cost range cameras and setups used in literature for color and depth data acquisition for hand gesture recognition purposes, highlighting their capabilities and limitations. The methods employed for respectively detecting the hand in the framed scene and segmenting it in its relevant parts are then analyzed with

a higher level of detail. The algorithm first exploits skin color information and geometrical considerations for discarding the background samples, then it reliably detects the palm and the finger regions, and removes the forearm. For the palm detection, the method fits the largest circle inscribed in the palm region or, in a more advanced version, an ellipse.

A set of robust color and geometric features which can be extracted from the fingers and palm regions, previously segmented, is then illustrated accurately. Geometric features describe properties of the hand contour from its curvature variations, the distances in the 3D space or in the image plane of its points from the hand center or from the palm, or extract relevant information from the palm morphology and from the empty space in the hand convex hull. Color features exploit, instead, the histogram of oriented gradients (HOG), local phase quantization (LPQ) and local ternary patterns (LTP) algorithms to provide further helpful cues from the hand texture and the depth map treated as a grayscale image. Additional features extracted from the Leap Motion data complete the gesture characterization for a more reliable recognition. Moreover, the thesis also reports a novel approach jointly exploiting the geometric data provided by the Leap Motion and the depth data from a range camera for extracting the same depth features with a significantly lower computational effort.

This work then addresses the delicate problem of constructing a robust gesture recognition model from the features previously described, using multi-class Support Vector Machines, Random Forests or more powerful ensembles of classifiers. Feature selection techniques, designed to detect the smallest subset of features that allow to train a leaner classification model without a significant accuracy loss, are also considered.

The proposed recognition method, tested on subsets of the American Sign Language and experimentally validated, reported very high accuracies. The results showed also how higher accuracies are obtainable by combining proper sets of complementary features and using ensembles of classifiers. Moreover, it is worth noticing that the proposed approach is not sensor dependent, that is, the recognition algorithm is not bound to a specific sensor or technology adopted for the depth data acquisition. Eventually, the gesture recognition algorithm is able to run in real-time even in absence of a thorough optimization, and may be easily extended in a near future with novel descriptors and the support for dynamic gestures.

Abstract

La recente introduzione di applicazioni 3D e monitor stereoscopici ha creato la necessità di nuove interfacce uomo-macchina. I classici dispositivi di input, come la tastiera e il mouse, non sono in grado di sfruttare appieno il potenziale di queste interfacce e non offrono un'interazione naturale. I gesti, invece, forniscono un modo più naturale e sicuro di interagire con computer e altre macchine senza doverle toccare. I campi d'applicazione per le interfacce basate sui gesti spaziano dai videogiochi al riconoscimento automatico del linguaggio dei segni, all'assistenza sanitaria, alla robotica e all'automatizzazione dei veicoli.Il riconoscimento automatico dei segni è un problema impegnativo che sta interessando la comunità scientifica da diversi anni grazie alla sua applicabilità alle interfacce naturali. I primi metodi, basati sul riconoscimento a partire da immagini o video, erano affetti dai tipici problemi che caratterizzano questo tipo di dati. Inter-occlusioni, diverso colore della pelle anche tra utenti della stessa etnia e condizioni di illuminazione instabili, infatti, hanno spesso reso questo problema intrattabile. Altri metodi, invece, hanno risolto i problemi precedenti obbligando l'utente a indossare guanti sensorizzati o ad afferrare strumenti progettati per favorire la localizzazione della mano nella scena.La recente introduzione nel mercato consumer di nuovi sensori di profondità a basso costo, come il Kinect di Microsoft, lo XTION di Asus, il Senz3D di Creative, e il Leap motion, ha aperto la strada a metodi di riconoscimento dei gesti innovativi che sfruttano l'informazione sulla geometria della scena. La maggior parte dei metodi condivide una pipeline di riconoscimento comune basata prima sull'identificazione della mano nella scena, poi nell'estrazione di opportuni descrittori dai campioni della mano e infine nell'utilizzo di opportune tecniche di apprendimento automatico per riconoscere il gesto eseguito all'interno di un "dizionario dei gesti" predefinito.Questa tesi, basata sul fondamento precedente, propone un nuovo sistema di riconoscimento dei gesti che sfrutti descrittori sia sul colore sia sulla geometria della scena estratti dai dati provenienti da un sensore di profondità a basso costo. La tesi comincia con l'introduzione del problema del riconoscimento automatico dei gesti, mostrando una panoramica sugli algoritmi allo stato dell'arte e sulla filiera di riconoscimento adottata. Poi, la tesi descrive brevemente i sensori di profondità a basso costo principali e i sistemi usati in letteratura per l'acquisizione di informazioni sul colore e sulla profondità per scopi di riconoscimento dei gesti, evidenziando le loro potenzialità e i loro limiti. In seguito la tesi

analizza con maggiore dettaglio i metodi impiegati rispettivamente per la localizzazione della mano nella scena ripresa e la sua segmentazione nelle parti rilevanti. L'algoritmo prima sfrutta l'informazione sul colore della pelle e alcune considerazioni sulla geometria della mano per rimuovere i campioni riferiti allo sfondo, poi localizza accuratamente le regioni del palmo e delle dita e rimuove la regione del braccio. Per la localizzazione del palmo, il metodo fitta il più grande cerchio inscrivibile nella regione del palmo o un'ellisse.Un insieme di feature robuste sul colore e sulla geometria che possono essere estratte dalle regioni del palmo e delle dita, segmentate in precedenza, è poi descritto con accuratezza. Le feature sulla geometria descrivono proprietà del bordo della mano come le sue variazioni di curvatura, le distanze nello spazio 3D o nel piano immagine dei suoi punti dal centro della mano o dal palmo, o estraggono informazioni rilevanti sulla morfologia del palmo e dagli spazi vuoti nel suo guscio convesso. Le feature sul colore sfruttano, invece, gli algoritmi histogram of oriented gradients (HOG), local phase quantization (LPQ) e local ternary patterns (LTP) per ottenere altre informazioni rilevanti sulla tessitura della mano o sulla mappa di profondità trattata come un'immagine in scala di grigi. Feature aggiuntive estratte dai dati provenienti dal Leap Motion completano la caratterizzazione dei gesti per un riconoscimento più affidabile. Inoltre, la tesi descrive anche un nuovo approccio che sfrutta unitamente i dati sulla geometria provenienti dal Leap Motion e quelli sulla profondità provenienti da un sensore di profondità per l'estrazione degli stessi descrittori della profondità con un impegno computazionale inferiore. Questo lavoro in seguito affronta il delicato problema della costruzione di un modello di riconoscimento dei gesti robusto dalle feature descritte in precedenza, usando Support Vector Machines, Random Forests o più potenti insiemi di classificatori. Sono anche considerate tecniche di selezione delle feature per rilevare il minor sotto insieme di feature che permetta l'allenamento di un modello di classificazione senza una significativa perdita di accuratezza.Il metodo di riconoscimento dei gesti proposto, testato su sotto insiemi di segni dell'alfabeto American Sign Language e validato su dati reali, ha riportato accuratezze molto elevate. I risultati hanno anche mostrato che le accuratezze maggiori sono ottenibili con la combinazione di opportuni insiemi di feature complementari e usando insiemi di classificatori. Inoltre, è opportuno notare che l'algoritmo di riconoscimento non è legato a uno specifico sensore o tecnologia adottata per l'acquisizione di dati di profondità. Infine, l'algoritmo di riconoscimento dei gesti può essere eseguito in tempo reale anche in assenza di una completa ottimizzazione, e può essere esteso facilmente in un prossimo futuro con nuovi descrittori e con il supporto per i gesti dinamici.

Contents

List of Figures

List of Tables

List of Equations

List of Algorithms

Acknowledgments

It has been a long journey in space and time that, as many important experiences in life, enriched both my mind and my spirit. I was not, however, alone, since several valuable people walked by my side. I owe them a lot, surely more than a simple thank. First of all, I want to thank my advisor, prof. Guido M. Cortelazzo, for inviting me to enter his research group and for his support since the first moment. I want also to thank all the researchers, PhD candidates, graduates and undergraduates who gave a significant contribution with their research to the work in this thesis: Pietro Zanuttigh, Loris Nanni, Mauro Donadeo, Mauro Piazza, Mariano di Noia, Ludovico Minto, Giulio Marin, Marco Fraccaro, Saverio Zamprogno and Carlo Dal Mutto. In particular, Pietro Zanuttigh deserves a special mention for his guidance and the valuable help he gave me in this research. Finally, I want to thank my family, my friends and the personnel of the Department of Information Engineering of Padua for their support in all these years, and all the people I may have omitted in these few words and to whom I owe an acknowledgment.

To prof. Guido M. Cortelazzo,
for his friendship, guidance and support.

Chapter 1

Introduction

Nowadays 3D applications and games are widely populating personal computers, notebooks and tablets, favored from the rapid development of powerful CPUs and 3D graphics accelerators offered at an affordable price.

The rapid development of 3D applications and technologies has created the necessity of novel human-machine interfaces that may no longer be easily driven by the traditional input devices, such as keyboard and mouse, born for a non natural interaction with flat 2D environments.

Hand gestures provide, instead, a more natural and sometimes safer way of interacting with computers and other devices without touching them. After all, touch-less interfaces driven by hand gestures may be considered the next step in the human-machine interfaces evolutionary scale started with an uncomfortable interaction with keyboards, quickly simplified by the introduction of the mouse, and now completely replaced by highly intuitive finger taps on touch screens. The use cases for gesture-based interfaces range from the entertainment field to many other aspects of the daily life. The first key application is gaming, where hand gestures allow the user a more thorough and straightforward interaction with personal computers and other gaming platforms. Another key application is automatic sign-language interpretation, that would allow hearing and speech impaired people to interact with computers and other electronic devices. Health care is another field which may benefit from the usage of hand gesture recognition interfaces. In fact, not only hand gestures offer the surgeons a more natural consultation of diagnostic data such as 3D tomographies, but also the remote control of surgical devices in aseptic environments. Moreover, 3D data visualization in general is now an important requirement in several research fields. Hand gestures provide a more natural interaction also with recent humanoid robots, which aim at mimicking the human movements and behavior, as gestures may be associated

to commands the robot has to execute. Automotive industry is interested as well in hand gestures, in order to release in a near future novel interfaces which offer the user a more natural interaction with the dashboard and the board computer. Finally, one of the most important applications of hand gesture is the realization of human-machine natural interfaces. Hand gestures may be used, in facts, to replace the mouse in computer interfaces and to allow a more natural interaction with mobile devices like smartphones and tablets, but also with newer wearable devices like the Google glasses. Besides controlling standard 2D interfaces, a very interesting field is the interaction with 3D virtual environments, that is much more natural if the gestures are performed in the 3D space without using any control device or even touching the screen [2] (e.g., Fig. 1.1). 3D visualization of virtual environments is now possible also for consumers thanks to the recent introduction in the mass market of stereoscopic display technologies that has boosted the diffusion of 3D movies and other multimedia contents initially only accessible in cinemas. Nvidia, for example, developed a complete 3D vision environment for its graphics accelerators, employing active goggles and monitors for offering the user a more engaging gaming experience. Acer and LG produced active 3D monitors compatible with Nvidia's solution, while HP offered a more affordable passive 3D monitor also compatible with AMD graphics accelerators and able, thanks to the TriDef Ignition driver, to leverage the native 3D capabilities of games and applications powered by Microsoft's DirectX APIs.

Figure 1.1: Example of 3D interface driven by hand gestures from a movie

Hand gesture recognition, namely the task of automatically recognize the gesture performed by a person (or *user* in this context) selected from a predefined "gesture dictionary", requires to track the position and orientation changes of a user's hand and of the fingers moving in the 3D space. A simpler 3D extension

of ordinary 2D mouse capabilities may just require the tracking of one finger (e.g., the index finger), while more complex gestures may also need the estimation of the position and orientation of the finger tips and the phalanxes. This problem has been attaining a growing interest in the research field for several years due to its applications in natural interfaces. The first approaches, based on the recognition from 2D color pictures or video only, suffered of the typical problems characterizing such type of data. Inter occlusions, different skin colors among users even of the same ethnic group and unstable illumination conditions, in fact, often made this problem intractable. Other approaches, instead, solved the previous problems by making the user wear sensorized gloves or hold proper tools designed to help the hand localization in the scene, thus renouncing to the naturalness of the the interaction.

The recent introduction in the mass market of novel low-cost range cameras, like the Microsoft Kinect^{TM}, Asus XTION and Creative Senz3D, and the Leap Motion, has opened the way to innovative gesture recognition approaches exploiting the geometry of the framed scene. Most methods share a common gesture recognition pipeline based on firstly identifying the hand in the framed scene, then extracting some relevant features on the hand samples and eventually exploiting suitable machine learning techniques in order to recognize the performed gesture from a predefined gesture dictionary.

This thesis, based on the same rationale, proposes a novel gesture recognition framework (named "HAndy") exploiting both color and depth cues from low-cost color and range cameras, and is articulated as follows. The remaining sections of the current chapter introduce the automatic hand gesture recognition problem, giving an overview of the state-of-art algorithms, and the recognition pipeline employed in this work. Chapter 2 briefly describes the current major technologies for the acquisition of color and geometric information, focusing on the low-cost range cameras used in literature for hand gesture recognition purposes and highlighting their capabilities and limits. Chapters 3 and 4 describe with a higher level of detail the method employed for respectively detecting the hand in the framed scene and segmenting it in its relevant parts. Chapter 5 then analyzes accurately several feature descriptors that can be extracted from the segmented fingers and palm samples, each one describing significant geometric or textural properties of the hand. Chapter 6 addresses the delicate problem of constructing a robust gesture recognition model from the features of Chapter 5, using multi-class Support Vector Machines, Random Forests or more powerful ensembles of classifiers. It also considers a few feature selection techniques, de-

signed to detect the smallest subset of features that allow the training of a leaner classification model without a significant accuracy loss. Chapter 7 reports and discusses the results of several tests performed on subsets of the American Sign Language alphabet to measure the computational and recognition performance of the proposed algorithm. Chapter 8, finally, draws the conclusions.

1.1 Problem definition

Most automatic gesture recognition approaches share a common pipeline, depicted in Fig.1.2, with minor variations.

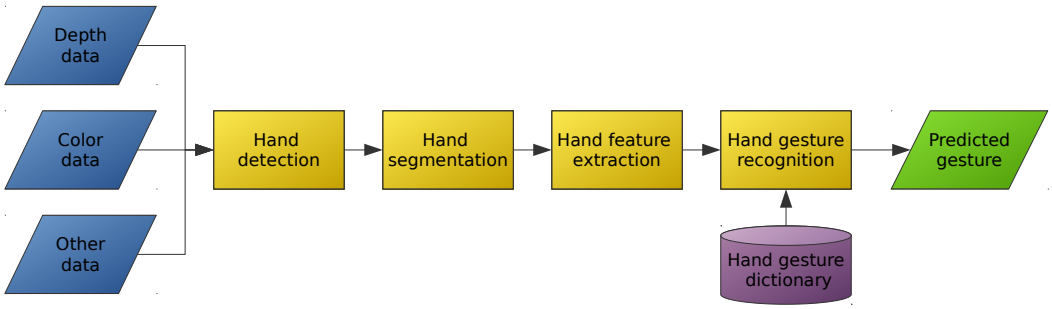
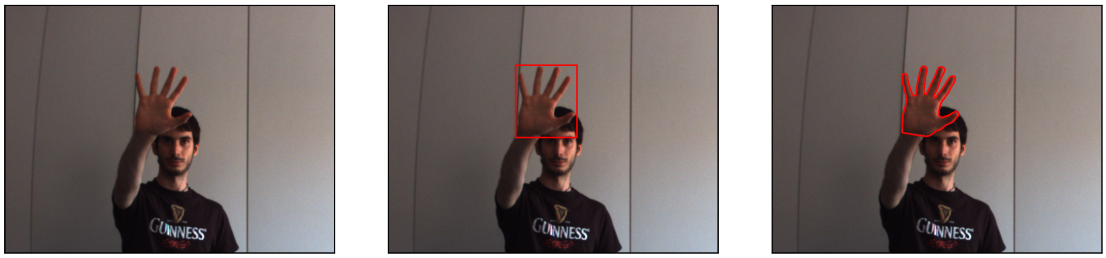


Figure 1.2: General pipeline of an automatic hand gesture recognition algorithm.

The first step of the pipeline of Fig. 1.2 is the hand *detection*, namely finding a region in the color image or in the depth map where the hand is likely to be located. This task may be divided into two parts: in the first part the rough hand location is estimated (see Fig. 1.3(b)), then in the second one the hand pixels are precisely isolated from the rest of the scene (Fig. 1.3(c)). In the first part the target is usually the identification of a bidimensional (or three dimensional if depth information is used) bounding box containing the hand, while in the second part the goal consists in removing the remaining background pixels.



(a) Acquired color image (b) Detected hand region (c) Segmented hand

Figure 1.3: Example of hand detection from depth and color data

This step is important for two reasons: first, the bounding box position gives a good starting point for a further accurate hand region segmentation, second, restricting the region of interest may sensibly reduce the computational load in the next phases. Hand segmentation algorithms may be based on the exploitation of color information, edge information, graph-connectivity information between pixels, and 3D geometry information if available. Moreover, some priors on the hand shape may be considered in order to solve this task.

After detecting the hand in the framed scene, the segmentation step partitions the hand pixels in different subsets referred to hand regions of interest, e.g. palm and fingers, each one containing relevant information about the hand orientation and finger opening status.

Feature extraction step is one of the most crucial phases in every pattern recognition method, and consists in extracting a set of relevant features from the hand only samples describing properties of interest. The features must be *robust*, in the sense they must contain useful information for the gesture recognition purpose, and *repeatable*, namely they must assume similar values to equality of conditions. Another desirable property for effective features regards their *correlation*: the selected features should not be mutually correlated, as highly correlated features are not informative.

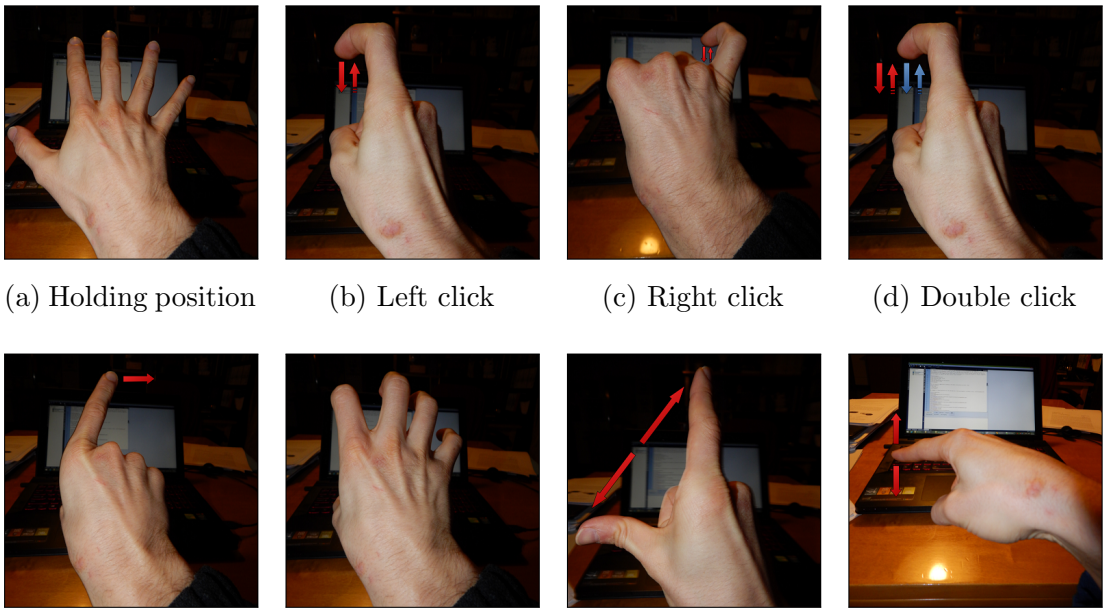
The extracted features are finally collected and used as the input of a suitable machine learning technique in order to recognize the performed gesture from a predefined "gesture dictionary". The dictionary size and contents depend on the application where the gestures are employed. For example in [2], where the hand is thought as a replacement of the ordinary 2D mouse for a more natural navigation in 3D environments, the base dictionary should include firstly all the gestures corresponding to the common actions that can be performed by a classical 2D mouse on desktop computers, and by fingers on tablets and smart-phones touch screens:

- holding position
- left and right click
- double-click
- 2D translation of the pointer
- 2D drag & drop

This set may be completed by adding a few more articulated actions, like:

- zoom
- scroll

An example of hand gestures for the navigation in 2D environments is shown in Fig. 1.4.



(a) Holding position (b) Left click (c) Right click (d) Double click

(e) Translation (f) Drag & drop (g) Zoom (h) Scroll

Figure 1.4: Example of typical gestures to be recognized in a 2D interface

However, for the purposes of 3D scene browsing this dictionary is quite limited and does not exploit the extra features provided by hand gesture interfaces, namely the localization of the position in 3D space instead of 2D image plane and the the multi-finger movements. For this reason, the base set should be extended by adding the following gestures:

- 3D translation of the pointer
- 3D drag& drop
- 3D rotation of an object that is hold under clicking
- 3D rototranslation of an object that is hold under clicking
- articulated multi-finger gestures in 3D

For automatic sign interpretation purposes, instead, the dictionary should include all the gestures representing the letters or symbols of the adopted language alphabet, such as the American Sign Language alphabet (Fig. 1.5).

It is worth noting that, while several approaches in literature share the same initial steps, they usually differ in the extracted feature sets or the employed classifiers. For this reason, the accuracy of different methods applied to the same data mostly depends from the types of extracted features and/or from the classification algorithm used to train the gesture recognition model.

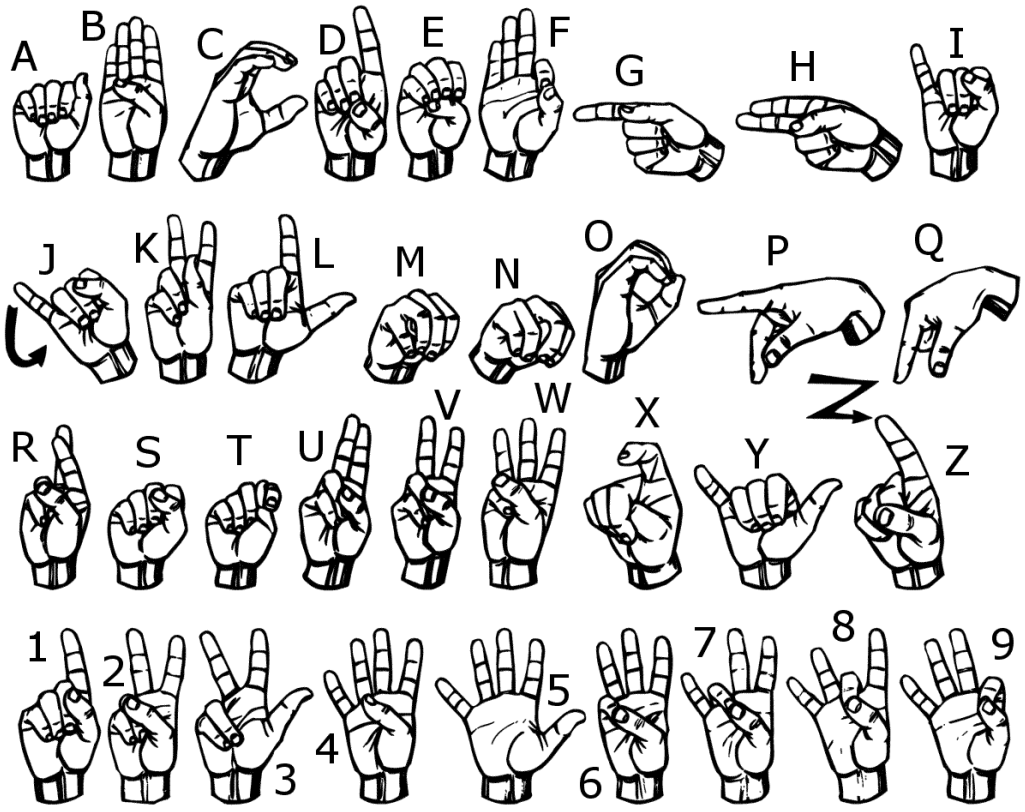


Figure 1.5: Gestures from the American Sign Language alphabet

1.2 Related works

Automatic gesture recognition is a challenging problem that has been attaining a growing interest in the research field for several years due to its applications in natural interfaces, as reported in recent surveys (e.g. [3, 4]).

Until a few years ago all the available methods were based on the extraction of color or motion information from images and videos, or exploited the shape of the hand silhouette. A complete overview of these approaches is out of the scope of this thesis, which focuses mostly on depth information, and may be found in [3] and in [5]. Gesture recognition methods based on color cues only suffered from the typical problems characterizing color data: multiple hand poses often presenting inter occlusions between fingers, different skin colors among users even of the same ethnic group and different illumination conditions, in fact, often made this problem intractable.

Other approaches, instead, solved the previous problems by making the user wear sensorized gloves, such as the 5DTGlove of 5DT (Fig. 1.6(a)), or hold proper tools designed to help the hand localization in the scene. Sony and Nintendo, for example, solved the hand tracking problem with their Playstation Move and Wiimote respectively (Fig. 1.6(b) and Fig. 1.6(c)). Even though gloves and various wearable devices have been used in the past, vision-based methods are nowadays preferred as they are able to capture the hand gestures without requiring the user to wear any physical device, thus allowing a more natural interaction with computers and many other devices.

(a) 5DT 5DTGlove

(b) Sony Playstation Move

(c) Nintendo Wiimote

Figure 1.6: Example of commercial hand detection aid devices

The introduction in industry of Time-Of-Flight cameras, such as the SR4000 and the newer SR4500 of Mesa Imaging, and more recently in the mass market of novel low-cost range cameras like the Microsoft Kinect™, Asus XTION and Creative Senz3D, has opened the way to innovative solutions for several challenging

computer vision problems exploiting the geometry of the framed scene, including object tracking and recognition, human activity analysis, indoor 3D mapping and also hand gesture recognition. A complete review of them is presented in [6]. In particular the success of Microsoft's Kinect^{TM}, initially designed for tracking the body movements for gaming purposes but early adopted by researchers as well, has shown how natural interfaces based on the acquisition of 3D data can be efficiently employed in commercial applications. Face detection is another computer vision task which benefited from the geometric cues provided by the low-cost range cameras. The work of [7], for example, improves the viola-jones face detector [8] discarding the face candidates with an incompatible head size or detected on a flat surface (e.g., a poster).

Along with range cameras, the growing interest in gesture based interfaces led to the recent introduction in the mass market of the Leap Motion sensor, a device explicitly targeted to hand gesture recognition that, differently from the low-cost range cameras that allow to obtain a complete 3D description of the framed scene, it only provides a limited set of relevant points.

Several automatic gesture recognition methods, as stated in Section 1.1, follow the general pipeline depicted in Fig. 1.2, adopted by many other pattern recognition algorithms as well. The first important step is hand detection, namely the discovery of a region in the framed scene where the hand is most likely to be located, followed by the background removal.

Earlier approaches mostly used skin color only as a detection cue (e.g., [9]), retaining pixels whose colors are considered to be in the skin color range and discarding the others. These methods were more likely to fail, since there are several issues regarding color to deal with: for example, skin color varies among people of different ethnic groups or even in the same ethnic group, and it is not uniform among hand pixels of the same person as well. Moreover, colors are affected by the scene illumination and the images may also contain also other bare body parts (e.g. arms, face) or other objects sharing the same skin color color range. Note how most of the hand detection from skin color approaches exploit cascade classifiers based on Haar-like features [8] employed for face detection. However, differently from faces having fixed properties related to the position of the mouth, eyes and nose, hands have many degrees of freedom, thus making this technique not effective for the hand detection case.

Other hand detectors, instead of color evaluation or together with it, employ motion or silhouettes as detection clues. In particular, silhouette detectors compare the object shape with any possible hand shape (or template) and discard

the region if the shapes do not match. A first observation is that the hand is usually the closest object to the acquisition setup (as in the example of Fig. 1.3) and if depth information is available, the simplest detection strategy only consists in performing an appropriate filtering on the depth values ([10, 11]). Additional geometric constraints in the hand aspect ratio and size may be used to refine the segmentation, as in [12].

Other approaches exploiting depth information only, use clustering algorithms such as K-means, iterative seed fill or region growing to separate the hand region from the rest of the scene. In [13], the depth range is fixed and a flood fill algorithm is used to cluster contiguous points with the aim of separating the hand from the body. In [14], instead, the K-means algorithm with two clusters is used in a limited depth range to find the hands. Note that when the hand shares its depth range with other objects, the hand detection by a simple depth filtering fails, and more information is required to effectively segment the hand from the background. The assumption that the hand has to be the closest object in the scene can be relaxed by predicting the hand depth according to the position of other body parts, such as the face.

In addition, when the color image is available as well, skin color segmentation can be used to enforce the hand detection. In [15], skin color segmentation based on both a model trained offline and a further online histogram-based refinement are used to obtain an initial guess of the hand position. Then, the user face is detected and all the points not belonging to a predefined region in front of the face are rejected. Once the hand is detected, the arm is removed by exploiting the depth and other geometrical constraints. Joining color and depth information may give another advantage: depth filtering discards objects or parts of them not included in the hand depth range, and color filtering can be used to discriminate the hand among objects within its depth range. Other clues may be joined to color and depth, if available, paying a little overhead in the detection algorithm performance.

Other approaches also exploit some physical aids, e.g. in [11] the users have to wear a black bracelet around the gesturing hand wrist to help the hand detection in the color image after a depth thresholding.

Finally, more reliable methods exploit the temporal redundancy to better find and segment the hand, reducing the false positive detection. For example, [16] first divide depth map into a given number of blobs using a connected-component labeling algorithm, then, for the biggest blob (that is assumed to include the body and the hand), compute blob tracking. The blob with the highest track is

associated to the hand. Additional geometric constraints are also used to identify and remove points of the wrist region.

Hand segmentation then divides the detected hand in its region of interest for the extracted features in the following important step. Certain features are extracted from the whole hand data, while others only from a limited subset (e.g., palm or finger regions).

Feature extraction is a fundamental step which often determines the success in recognizing gestures. Methods working on the same data set may, in fact, show relevant differences in recognition accuracy due to the kinds of extracted features or in the implementation of the extraction algorithms.

A first family of approaches is based on the hand silhouette extracted from the depth data. Ren et al. [11, 17], for example, build an histogram of the distances of the hand contour points from the centroid of the palm. This approach is affected by the fact that the palm contour is also considered in the histogram construction. Better performance can be obtained if the palm and finger areas are recognized before building the histogram or other descriptors based on the contents of the two regions. In [16] silhouette and cell occupancy features are extracted from the depth map and used for building a shape descriptor that is then fed into a classifier based on action graphs. Other approaches in this family use features based on the convex hull and on the fingertips positions computed from the silhouette, as in [18] and [14]. The convex hull is also exploited in the open source library *XKin* of [19, 20]. [21] propose, instead, a simple application of hand gestures for human-robot natural interaction where the user challenges a robotic hand in the well-known rock-paper-scissors game. The gesture recognition algorithm exploits the hand contour *curvature* information to characterize the three gestures It is worth noting that the system of [21] is able to accurately recognize the three gestures performed with the only use of the bare hand, and an AI (artificial intelligence) tries to learn the user's gaming pattern in order to foresee his next moves.

Another possibility is computing descriptors based on the volume occupied by the hand. In [22], 3D volumetric shape descriptors are extracted from the depth map and fed into a Support Vector Machine (SVM) classifier [23]. A similar approach is exploited by [24]. Color data can also be used together with the depth data, as in [25], that is based on Randomized Decision Forests (RDFs) [26]. RDFs are also used by [27].

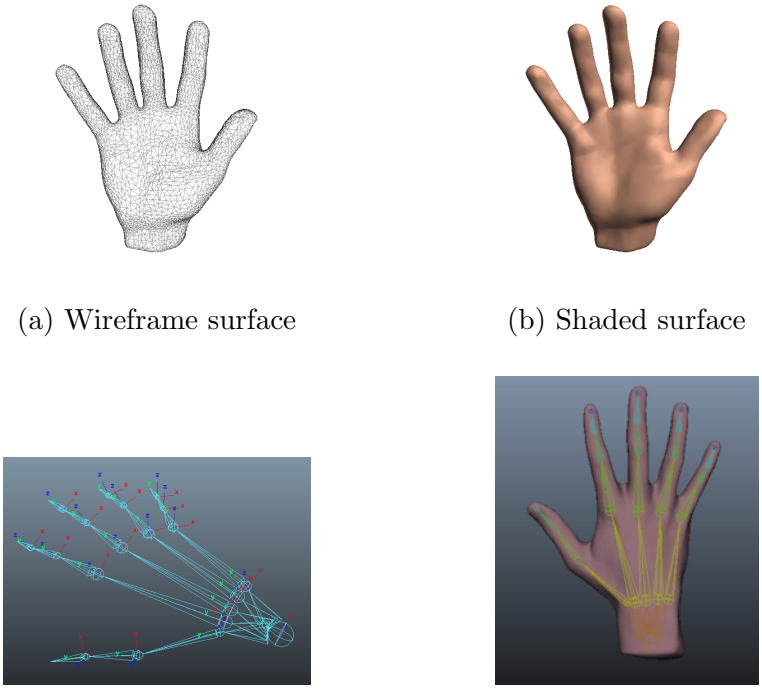
Note how all these approaches are focused on the recognition of static poses, while other methods, instead, deal with dynamic gestures. For example, [28]

exploit motion information, and in particular the trajectory of the hand centroid in the 3D space, for recognizing dynamic gestures. Depth and color information are used together in [29] to extract the trajectory that is then fed to a Dynamic Time Warping (DTW) algorithm. Finally, Wan et Al [30] exploit both the convex hull on a single frame and the trajectory of the gesture.

Among the various applications based on hand gesture recognition, sign language recognition is one of the most interesting. An approach for sign language recognition with the Kinect is proposed in [31]. A different but related problem is the extraction of the 3D hand pose, which can then be in turn exploited for gesture recognition. A first possible approach consists in fitting a parametric hand model to the acquired depth data, where the parameters model two different kinds of information: the shape information, that is the size and thickness of the various hand components, and the positional information, namely the position of the various components of the hand in the scene. According to the previous rationale, the hand is represented by a non-rigid surface (Fig. 1.7 (a)), that is a function of the shape of the user's hand, and of a kinematic skeleton defining the position of the various parts of the hand in the scene (Fig. 1.7 (c)). A deformable model (Fig. 1.7 (d)) may be developed similarly to the one proposed in [32], that is the current state-of-the-art in entire body modeling.

The next step is the estimation of the optimal values for its shape and position parameters, that is, the values minimizing the alignment error between the data obtained from the estimated parameters of the deformable model and the data acquired from the real scene by the camera and depth sensor according to a given norm (usually the L2 norm, for statistical reasons). A good cost function should account for many clues, including edges, silhouettes and 3D geometry information if available. A description of classical cost functions adopted in the case of human body tracking can be found in [33]. Moreover, as fingers movements in real human hands are limited, e.g. phalanxes may only revolve within a finite range around a local rotation axis, every model parameter may just adopt a finite range of values compatible to the allowed movements of hand part it represents. Hence, when using L2 norm and imposing such constraints, the optimization problem is configured as a constrained minimization in least squares sense.

Approaches exploiting depth data and skeleton fitting to a 3D hand model are [34], [35] and [36]. In particular [36] try to estimate the pose by segmenting the hand depth map into its different parts, with a variation of the machine learning approach used for full body tracking in [37]. Multi-view setups have also been used for this task by [35], since approaches based on a single camera



(a) Wireframe surface

(b) Shaded surface

(c) Kinematic skeleton

(d) Complete model

Figure 1.7: Example of deformable model for hand pose estimation

are affected by the large amount of occluded parts, making the pose estimation rather challenging.

Finally, differently from the Kinect, the exploitation of Leap Motion data for gesture recognition systems is still an almost unexplored field. A preliminary study on the usage of this device for sign language recognition has been presented in [38]. Another gesture interface based on the Leap Motion has been presented in [39], where the authors use the device to control a robot arm.

1.3 Proposed method overview

The gesture recognition framework proposed in this thesis, named *HAndy*, is the basis of the approach of [21] and of a possible natural interface based on gesture recognition for the interactive browsing of 3D scenes only relying on the bare hand. The system architecture, depicted in Fig. 1.8, extends the general pipeline of Fig. 1.2 adopted by the approaches resumed in Section 1.2. As already stated in Section 1.2, the overall structure of the gesture recognition algorithm shares several steps with the other methods in literature. In particular, the acquisition, hand detection, hand segmentation, feature extraction and classification macro

blocks in Fig. 1.8 are common to several hand gesture recognition approaches and other pattern recognition algorithms in general.

The system architecture in Fig. 1.8 encompasses three main steps. In the first step, described in Chapter 3, the hand samples collected with one of the acquisition systems of Chapter 2 are segmented from the background exploiting both depth and color cues from low-cost color and range cameras, and the additional information provided by the recent Leap Motion when available. It is worth noticing that, contrary to the assumption of [10, 11], the hand does not have to be the closest object to the sensor thanks to the skin color information or the Leap Motion data. The previous segmentation is then refined in Chapter 4 by further subdividing the hand samples into three non overlapping regions, collecting palm, fingers and wrist/arm samples respectively. The last region is discarded, since it does not contain information useful for gesture recognition. Note how the palm region detection and the main hand direction estimation play an important role both in the segmentation phase and in the next crucial step of the pipeline, analyzed in detail in Chapter 5 and consisting in the extraction of several features belonging to three families:

- **Depth features:** extracted from the acquired depth map, describe relevant properties of the hand contour in the 2D image plane or in the 3D space, and the palm morphology. For example, the distances of the contour points from the estimated palm center or the palm region (assumed to be flat), the convexities and concavities of the hand contour, the empty spaces in the convex hull enclosing the hand and the fingertip positions and orientations highly characterize each gesture.

- **Color features:** provide relevant information on the local variations of the hand texture in the neighborhood of the hand contour pixels.

- **Leap Motion features:** when available, the Leap Motion data provide in real-time useful information on the hand pose which may integrate the features extracted from the depth map or redefine them in a more efficient way.

The extracted features are then collected in feature vectors fed to proper multi-class classifiers, described in Chapter 6, for the further gesture recognition. Note how the proposed classifiers are designed to leverage the uncorrelation of the features within the same set and their complementarity between different sets in order to maximize the recognition accuracy.

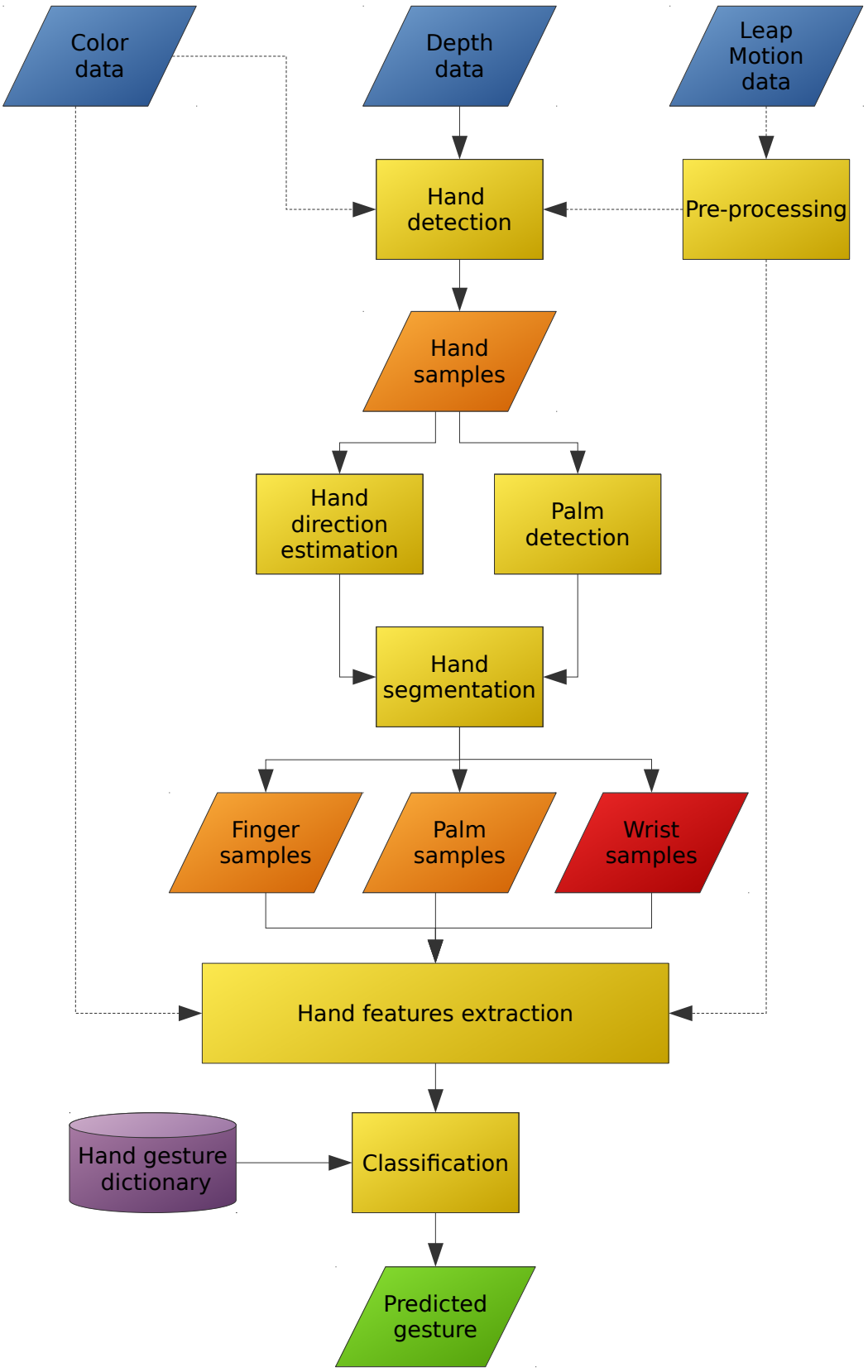


Figure 1.8: Architecture of the proposed gesture recognition framework

Chapter 2

Data acquisition

This chapter deals with the acquisition of color, depth and other kinds of information, a preliminary important step of the gesture recognition pipeline of Section 1.3. The choice of proper acquisition hardware and its arrangement in an acquisition setup is fundamental for the next phases, and depends both on the application and the budget. The quality of the data acquired by the selected setup may strongly affect the recognition accuracy, and it is often in a direct proportion with the hardware cost. Commercial applications developed for the mass market usually have to rely on robust gesture recognition algorithms able to compensate the inaccuracies of the data from inexpensive acquisition devices, while in other applications where an high accuracy is mandatory the usage of more expensive and accurate devices may not be avoidable.

The remainder of the current chapter describes briefly the main devices and acquisition setups employed in literature for gesture recognition purposes and for several computer vision tasks in general, highlighting their capabilities and limits and focusing on the most relevant low-cost depth sensors currently available in the mass market. A more thorough analysis of these devices and setups is beyond the scope of the thesis, and an exhaustive treatment can be found, instead, in [1]. It is worth noting that, as the proposed system aims at recognizing the gestures performed by the user's bare hand, any possible acquisition setup involving the usage of gloves or other tools compromising the naturalness of the gesture will not be considered.

Automatic hand gesture recognition and, generally, several other computer vision tasks exploiting color and depth or geometric cues, can use of one of the following devices or setups. Recall that, without a proper calibration of each device and especially of each multi-sensor setup, all the information collected will not lead to a correct hand pose and orientation estimation.

- one color camera only
- two color cameras (passive stereo system)
- two color cameras and a projector (active stereo system)
- one structured light device or system
- one range camera only
- one color camera and one range camera
- two color cameras and one range camera (trinocular system)
- one Leap Motion sensor only
- one Leap Motion sensor and a range camera or a stereoscopic system

2.1 Color cameras

Color cameras have been used for the solution of computer vision tasks since the birth of this research field, and they have the only available acquisition devices for several years. They are nowadays equipped as well to several personal computers and notebooks, each smartphone, tablet and other low-cost embedded devices, although their imaging sensor quality is generally poor for keeping the prices low.

Fig. 2.1 compares a few examples of color cameras available in the market, ranging from inexpensive cameras embedded in mobile devices to more professional models used in industry.

(a) Embedded camera (b) Computer webcam (c) Industrial camera

Figure 2.1: Example of currently available color cameras

The trade-off between the imaging sensor and lens quality and the overall device cost increments the chance of failure of earlier computer vision methods based on color data. Low-end smartphones, for example, are now often offered for not more than $100, but are equipped with imaging sensors poorly performing in low light conditions, while only the optics of a good quality camera are usually offered for hundreds dollars.

For this reason and for the depth information loss due to the color camera operation, most of the novel hand gesture recognition approaches reported in Section 1.2, including the work presented in this thesis, no longer make use of a single color camera as an acquisition setup.

Even though several recent gesture recognition approaches are no longer based on color data only, the camera model schematized in Fig. 2.2, that is the *pinhole* camera model, is still deeply entwined with almost all gesture recognition methods as it is the basis of several calibration protocols and the only way of linking the 2D points living in the image plane with the 3D world.

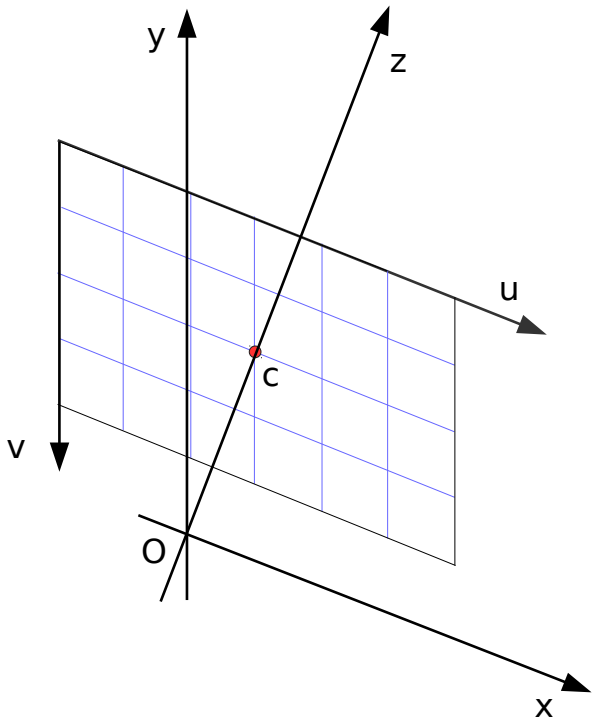


Figure 2.2: Pinhole camera model

Eq. 2.1 only reports the relevant equation of the pinhole model for notation purposes, according to the reference systems of Fig. 2.2. A more complete treatment can be found in [1].

$$z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\mathbf{X} = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.1}$$

where $[u \quad v \quad 1]^T$ are the homogeneous coordinates of the projection ray $\widetilde{\mathbf{p}}$ for the 3D point X with coordinates $\mathbf{X} = [x \quad y \quad z]^T$, K the intrinsic parameters matrix with f_u, f_v the focal lengths of the optics, c the image plane center with coordinates $\mathbf{c} = [c_u \quad c_v]^T$ and s the axis skew. Note how in most cases $s = 0$. From now on, for clarity sake, the projection of X on the image plane denoted by p with coordinates $\mathbf{p} = [u \quad v]^T$ will be considered in place of its homogeneous coordinates $\widetilde{\mathbf{p}}$.

2.2 Passive stereo setups

Passive stereo setups are a direct extension of the single camera ones, designed to overcome the third dimension (depth) loss of the latter due to the projection of 3D points on the image plane (Eq. 2.1). They are made by a pair of color cameras, usually of the same model and same intrinsic parameters, and their operation is exemplified in Fig. 2.3.

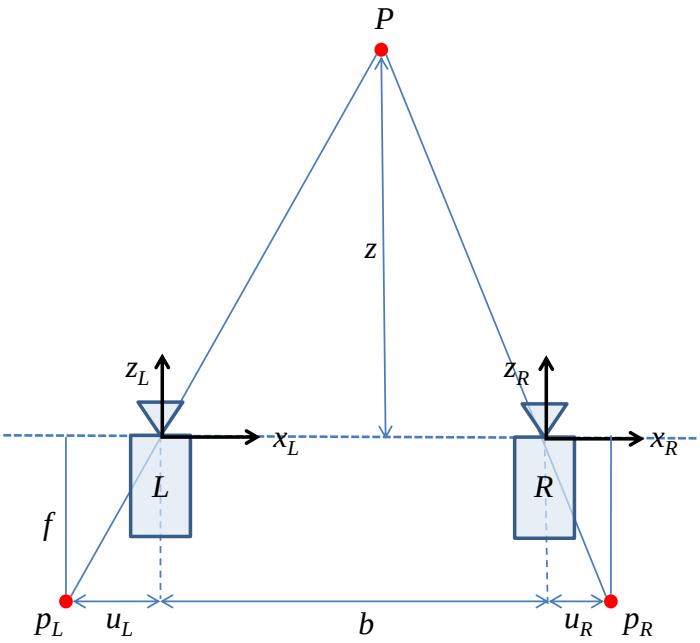


Figure 2.3: Stereo vision system operation (courtesy of [1])

According to the reference systems of Fig. 2.3, the depth z of the 3D point P can be computed by $z = b\frac{f}{d}$, where b is the *baseline*, namely the distance between the two optical centers, f the focal length of the optics and $d = (u_L - u_R)$ the *disparity* of the projection of P on the two camera image planes. Note how Fig. 2.3 represents an ideal system with identical cameras perfectly aligned, a situation hardly verified in real setups but approximable by using proper stereo calibration protocols [40].

Passive stereo setups are, again, affected by the image quality problems of single camera setups, and in addition by the accuracy of the 3D reconstruction algorithm adopted for the depth estimation. The accuracy of a 3D reconstruction method, in fact, depends on the reliability of the detected correspondences, which in turn strongly depends on the image quality. A more complete treatment of this topic can be found in [41].

Passive stereo setups are easy to build and to embed in small low-cost devices, as they are only made by a pair of color cameras but, as already stated, often the lower the device cost the lower the image quality and consequently the lower the number and the reliability of the detected correspondences. Moreover, note how the correspondence detection is a time-demanding task, and in order to obtain acceptable frame-rates several practical applications employing passive stereo systems have to rely on lower quality reconstructions to reduce the computational load.

Analogously to the single color camera setups, the market offers affordable and compact passive stereo cameras ranging from low-cost webcams (Fig. 2.4(a)) to more expensive solutions for professional applications (Fig. 2.4(b)). Finally several research groups often adopt self-made acquisition rigs (Fig. 2.4(c)).

(a) Low-cost stereo camera

(b) Professional stereo camera

(c) Ad-hoc stereo setup

Figure 2.4: Example of currently available passive stereo cameras and setups

2.3 Active stereo setups

Passive stereo rigs, as stated in Section 2.2, are easy to arrange and can be made of commodity cameras. While the theory behind their operation is rather simple, practical cases show that the reliable discovery of correspondences between a 3D point projected in the image planes of the two cameras is a challenging problem to solve.

Active stereo setups augment passive stereo with a projector casting a known pattern (e.g. a b/w checkerboard) on the framed scene with the aim of raising the number of correspondences in low-textured regions or smoothed surfaces. For example, PneumaCare developed the PneumaScan system (Fig. 2.5) for automatically measuring the of air into a patient's lungs over time.

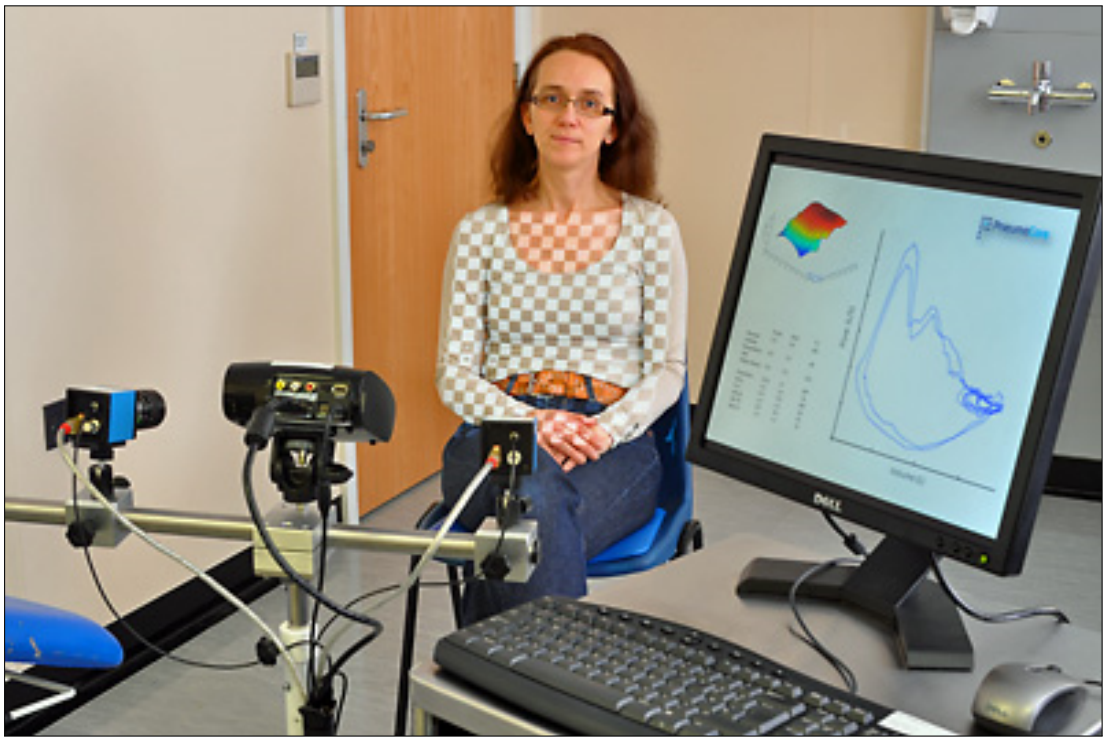


Figure 2.5: Active stereo rig for chest 3D surface reconstruction (PneumaCare).

PneumaScan system operates by first projecting a grid pattern from an LED-based digital light projector (DLP) on a patient's chest area, then two digital cameras capture the corner features of the checkerboard grid and two sets of 2D points are created from the images. By recording the changes in the projected pattern on a patient's chest in the two image planes, a dynamic 3D model of the chest can be generated. A software-based triangulation method then identifies each one of the grid locations and recreates a 3D representation of the chest

and abdominal wall surface. Changes in the chest volume are computed by the software from the reconstructed virtual surfaces and can be plotted graphically in real time as the patient breathes. Since the lungs are the only compressible part of the torso, the system can calculate the flow of air into the lungs, namely, how the volume of the torso changes over time.

Note how the introduction of a pattern projector both creates new correspondences and limits the capability of the system to be embeddable in a compact device. Moreover, when the projected light is in the visible spectrum the user's scene perception is altered.

2.4 Structured light sensors and setups

Structured light is an alternative technology to active stereo for the solution of the passive stereo correspondence detection problem. Such systems project known light patterns (usually in the infrared range) on the scene with a projector replacing one camera of a stereo system, and estimate the point depths from the pattern deformation (warping) on the image plane due to the scene geometry (Fig. 2.6).

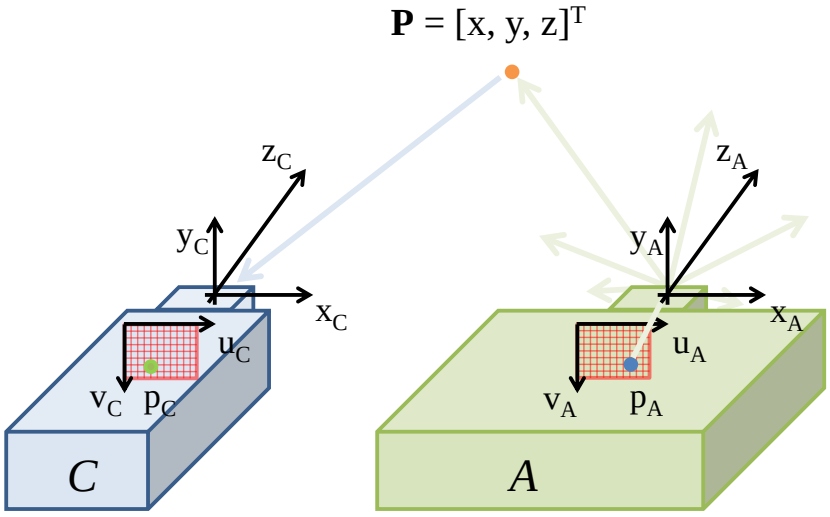


Figure 2.6: Structured light devices operation (courtesy of [1])

A first family of devices of this category is the one of laser scanners, widely used in industry for prototyping and in architecture or civil engineering to perform geometric measurements on the acquired 3D scene (eg., bridges, buildings). They are based on the controlled steering of one or more laser beams followed by a distance measurement at every pointing direction, as exemplified in Fig. 2.7.

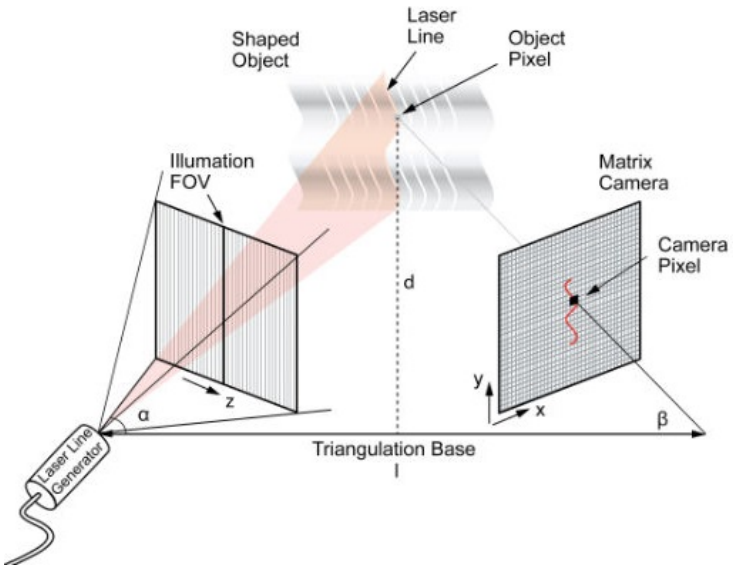


Figure 2.7: Laser scanner operation

As shown in Fig. 2.7, the projected light helps the detection of the correspondences since they are marked by high intensity points or curves in the image plane of the camera. Fig. 2.8 shows an example of laser scanners currently available in the market.

(a) Laser scanner for industry (b) Laser scanner for architecture

Figure 2.8: Example of currently available laser scanners

Although the achieved accuracies are extremely high (measurement error in the micrometer range), the long time required for each acquisition and the use of lasers make them unsuitable for the automatic gesture recognition purposes.

Other 3D scanners (Fig. 2.9) replace the laser generator with a video projector and exploit *time multiplexing* to rebuild the 3D shape of the framed scene. Each image pixel index is encoded with a unique light code (e.g., binary or gray code) made by alternating B/W patterns, usually dark stripes with width varying over time. Note how the usage of a second camera, like in the acquisition rig of Fig. 2.9, is not mandatory but reduces the lack of data due to inter occlusions.

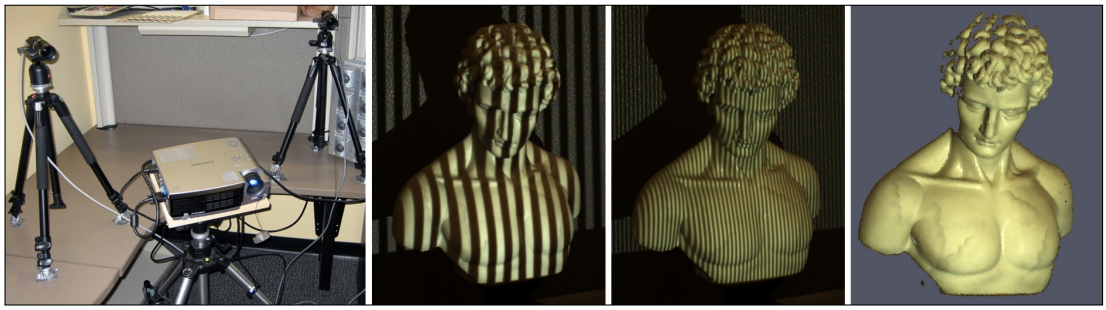


Figure 2.9: Example of 3D scanning based on structured light time multiplexing

While the accuracy achievable with time multiplexing techniques is, again, extremely high (around 40μ), the drawbacks are their narrowness to the acquisition of static scenes and the elevated number of generated patterns. Newer systems use, instead, colored light coding techniques (Fig. 2.10) able to achieve real-time performances with fewer frames (e.g. one or two) and a more sophisticated correspondence matching algorithm.

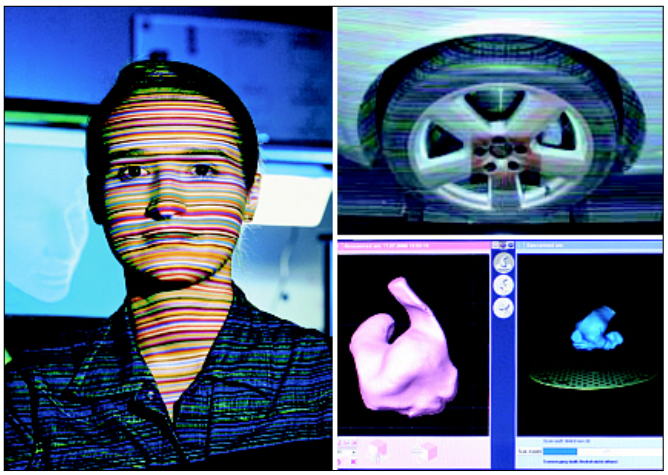


Figure 2.10: Example of 3D scanning based on structured colored light coding

Microsoft Kinect (ver. 1) and the Asus XTION are two low-cost structured light coding range cameras built upon the same PrimeSense system-on-chip (Fig. 2.11), whose operation is still undisclosed.

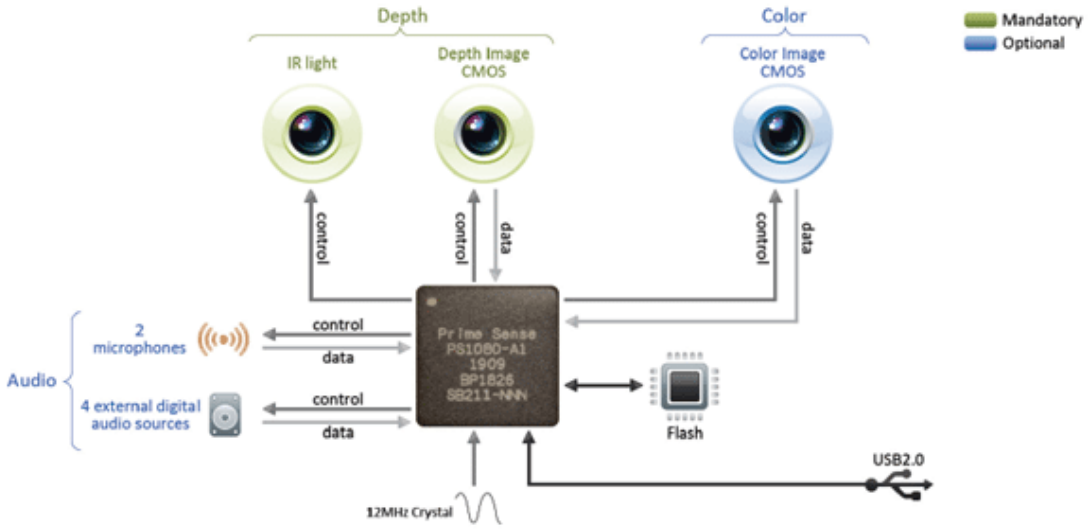


Figure 2.11: Architecture of the PrimeSense SoC

The Kinect was originally designed for gaming purposes as a Microsoft XBOX controller, but is now widely used in the computer vision field for its high depth map acquisition frame rate, while the XTION is expressly designed for personal computers. Both the devices are equipped with an infrared projector with a coupled infrared camera for the depth estimation, and a HD color camera for the color information acquisition.

A structured light infrared pattern (Fig. 2.12) is projected on the scene and its deformations by the scene geometry are captured by the infrared camera. Since the original pattern is known by the system and the "light codewords" are uniquely assigned to the pixels of the projector image plane, by tracking each codeword in the acquired infrared image it is possible to uniquely determine the correspondences between the infrared camera and the projector image plane and then estimate the depth from the pixel disparities by triangulation.

Note how the association of color and depth information to the same acquired sample requires a prior alignment of the color image with the depth map. Although such alignment could be automatically performed by old frameworks like OpenNI, better results may only be obtained by following ad hoc calibration protocols (e.g. [42]) different from the classic color camera calibration.

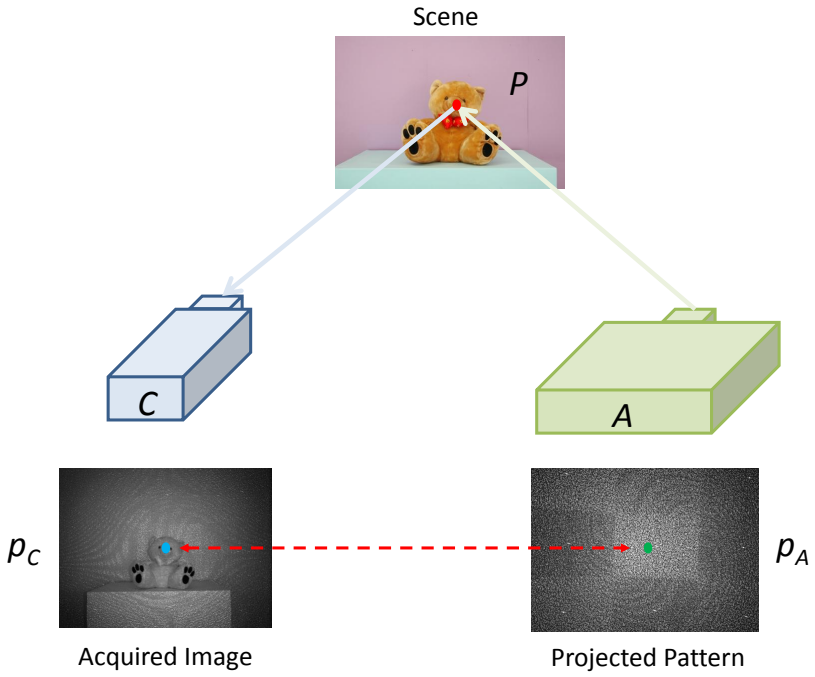


Figure 2.12: Structured light coding in Microsoft Kinect (courtesy of [1])

Finally, Occipital released Structure.io, targeted to IPad tablets.

(a) Microsoft Kinect ver.1 (b) Asus XTION (c) Structure.io

Figure 2.13: Example of available structured light coding low-cost range cameras.

2.5 Time-of-Flight cameras

Range cameras, such as Mesa SR4000 (Fig. 2.14(a)) for industry, the recent Microsoft Kinect 2^{TM}(Fig. 2.14(b)) and Creative Senz3D (Fig. 2.14(c)) for the mass market, implement an alternative technology to active or passive stereo and structured light systems that achieves higher frame rates with generally lower spatial resolutions [1, 43].

(a) Mesa Imaging SR4000 (b) Microsoft Kinect ver.2 (c) Creative Senz3D

Figure 2.14: Example of currently available ToF cameras.

While stereo systems use triangulation to reconstruct the scene geometry, range cameras are based on the *time-of-flight* principle exemplified by Fig. 2.15.

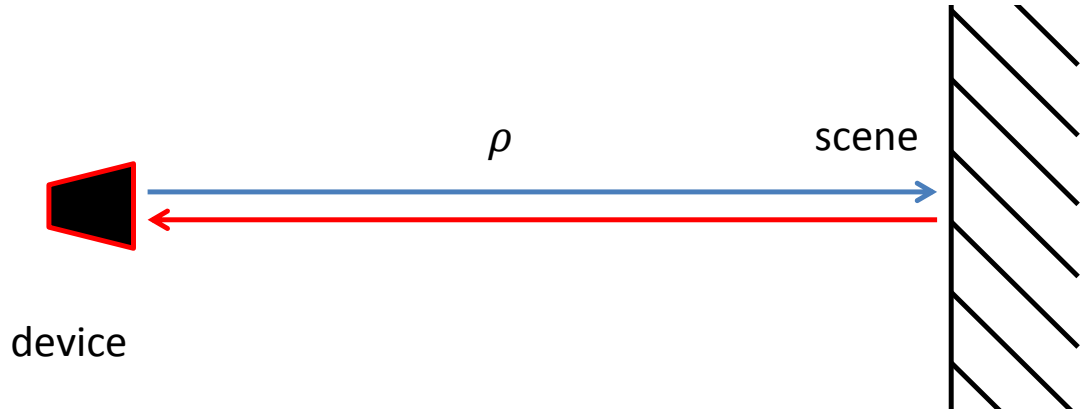


Figure 2.15: Time-of-flight principle (courtesy of [1])

Consider the single emitter-receiver pair of Fig. 2.15; the "time of flight" is defined as the round-trip-time of an infrared light pulse, that is, the time required by the pulse to hit the object at distance ρ from the emitter and being reflected back to the receiver. The rationale is that, as the light frequency is known a priori and the round-trip-time measurable, the object distance from the emitter (depth) can be computed by the simple Eq. 2.2.

$$\rho = c\frac{\tau}{2} \tag{2.2}$$

where c denotes the light speed in vacuum (although the actual speed of the infrared pulse is slightly lower) and τ the round-trip time of the infrared pulse.

It is worth noting that real range cameras are actually made by a grid of receivers collecting the infrared radiation from multiple emitters and carry out more complex computations than Eq. 2.2 to reliably estimate the geometry of the framed scene.

Furthermore, due to timing issues with light pulses, commercial range cameras often exploit the phase shift of sinusoidal waves as an indirect way of measuring the round-trip-time [1] (Fig. 2.16).

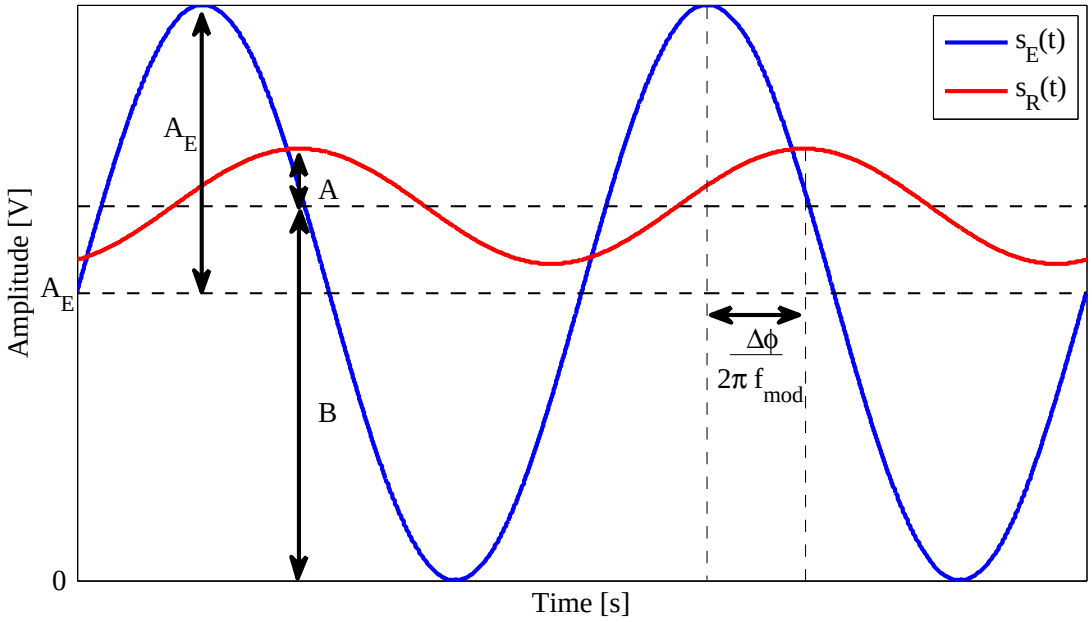


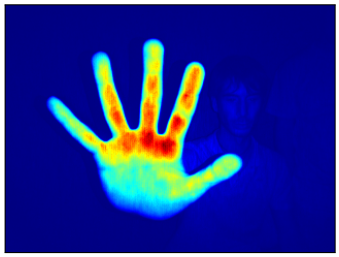
Figure 2.16: Continuous wave time-of-flight principle

The SR4000, the Senz3D and other tof cameras return, along with the acquired depth map, additional data useful for calibration or quality estimation purposes: an *intensity map* representing the average intensity of the reflected light collected by the sensor array, and a *confidence map* quantifying the reliability of the estimated depth samples. Some range cameras like Senz3D and Microsoft Kinect 2 also return a color image of the framed scene. An example of data described above is reported in Fig. 2.17.

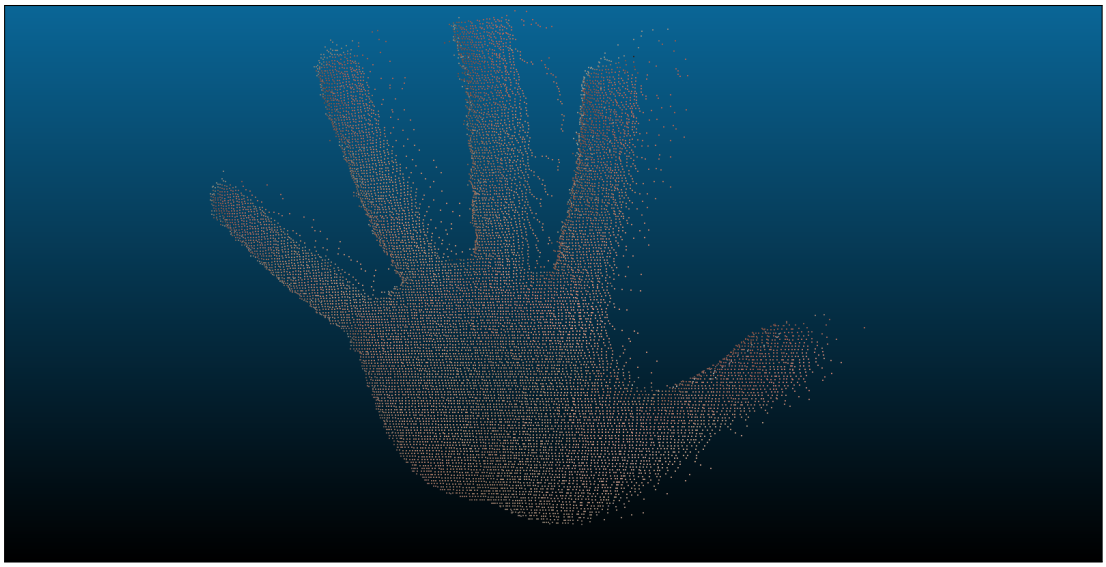
Finally, time-of-flight cameras usually provide better estimations of the scene geometry than low-cost structured light sensors like Microsoft Kinect (ver.1) and Asus XTION, but also require a rather high energy for empowering the infrared illuminators and do not perform well in presence of dark objects due to the absorption of most of the infrared radiation. Moreover, as they are highly sensible to direct sunlight, they should not be used in open environments, and they are generally not accurate along the object boundaries due to the "flying pixel" phenomenon caused by averaging the estimated depth along sharp edges (see Fig. 2.17(d)).

(a) Acquired color image

(b) Acquired depth map



(c) Acquired confidence



(d) Generated point cloud

Figure 2.17: Example of data returned by tof cameras

2.6 Binocular setup

This setup enables the acquisition of both color and depth information when using pure range sensors, like the SR4000 of Mesa Imaging, that only allow natively the collection of depth data. It is often used in the research field or in industry when available devices in the market are not suitable to the particular application.

Fig. 2.18 shows an example of acquisition rig made by an industrial range sensor and a professional color camera. Note how the two devices are placed one next to the other in order to minimize the reprojection error [1] and to simulate a unique camera collecting both color and depth information in the same imaging sensor. The acquisition rig needs an accurate calibration for correctly associating both color and depth information to the acquired scene samples.

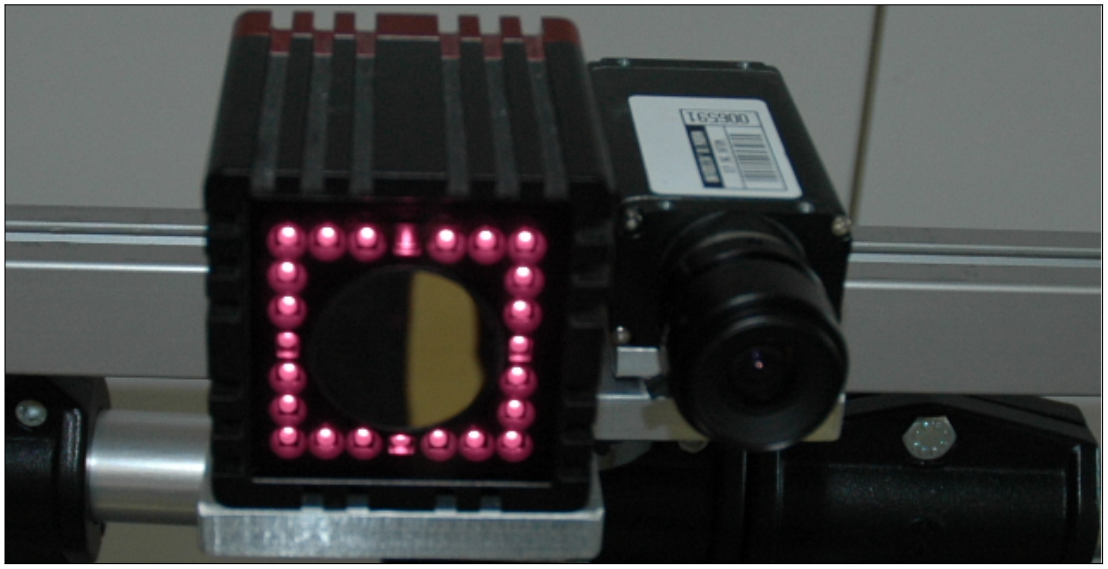


Figure 2.18: Example binocular acquisition rig made by a camera and a tof sensor

2.7 Trinocular setup

Trinocular setups are acquisition rigs made by a pair of matched color cameras (passive stereo) and a tof camera. They are designed to leverage the redundancy of the geometric data provided by both subsystems in order to compensate their inaccuracies by exploiting complex algorithms for depth data *fusion* [44]. The depth data from the range camera may, in fact, compensate the lack of correspondences when framing a uniform color wall and, conversely, the correspondences detected on a very dark surface may compensate for the low reliability of range data as stated in the end of Section 2.5. An example of trinocular setup is reported in Fig. 2.19 [1].

Trinocular systems, as any other multi-sensor setup, require an accurate ad-hoc calibration [44] in order to merge the depth data from the two subsystems.

2.8 Leap Motion

The Leap Motion is another recently introduced sensor based on vision techniques targeted to the extraction of 3D data for gesture recognition applications only. Differently from the depth acquisition devices or setups of the previous sections, that provide a complete 3D description of the framed scene, the Leap Motion produces a far more limited amount of information (only a few keypoints instead of the complete depth description) and works on a smaller 3D volume. On the

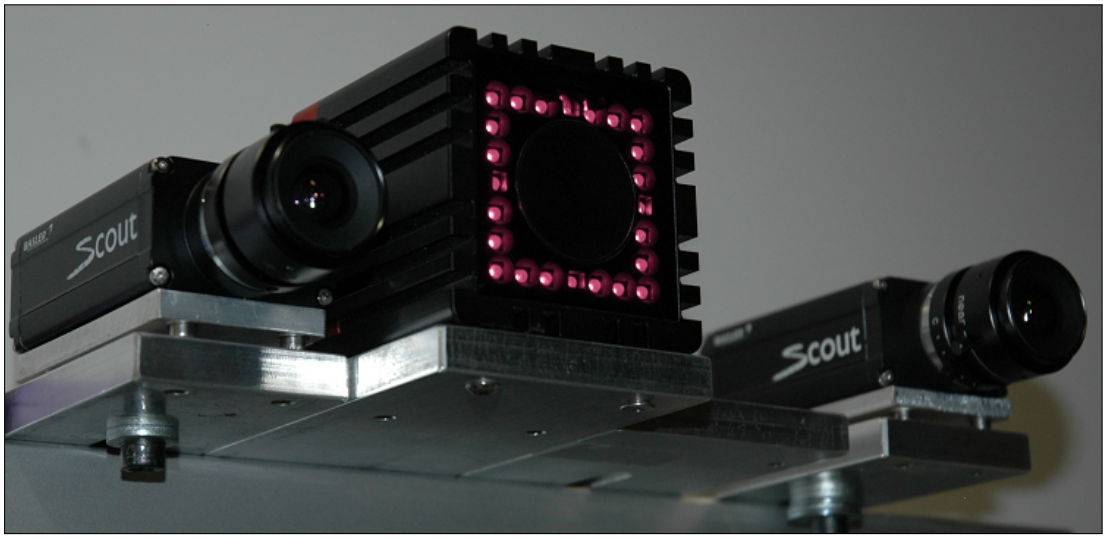


Figure 2.19: Trinocular acquisition setup made by a tof and two color cameras

other hand, the extracted data are more accurate (according to a recent study [45] the device accuracy is of about $200\mu m$) and it is not necessary to use computer vision algorithms to extract the relevant points since they are directly provided by the device software.

The first release of Leap Motion APIs only recognizes a few movement patterns, e.g., swipe or tap, and the exploitation of Leap Motion data for more complex gesture recognition systems is still an almost unexplored field. The sensor mainly provides the following data, depicted in Fig. 2.20.

- **Number of detected fingers:** N that the device is currently seeing.
- **Position of the fingertips:** F_i, $i = 1, \ldots, N$. Vectors $\mathbf{F}_i$ containing the 3D position of each of the detected fingertips. The sensor however does not provide a mapping between the vectors $\mathbf{F}_i$ and the fingers.
- **Palm center:** C that represents the 3D location roughly corresponding to the center of the palm region in the 3D space.
- **Hand orientation:** consists on two unit vectors representing the hand orientation computed in the palm center C. The first vector, denoted by $\mathbf{h}$, points from the palm center to the direction of the fingers, while the second, denoted by $\mathbf{n}$, is the normal to the plane that corresponds to the palm region pointing downward from the palm center.
- **Hand radius:** r is a scalar value corresponding to the radius of a sphere that roughly fits the hand curvature.

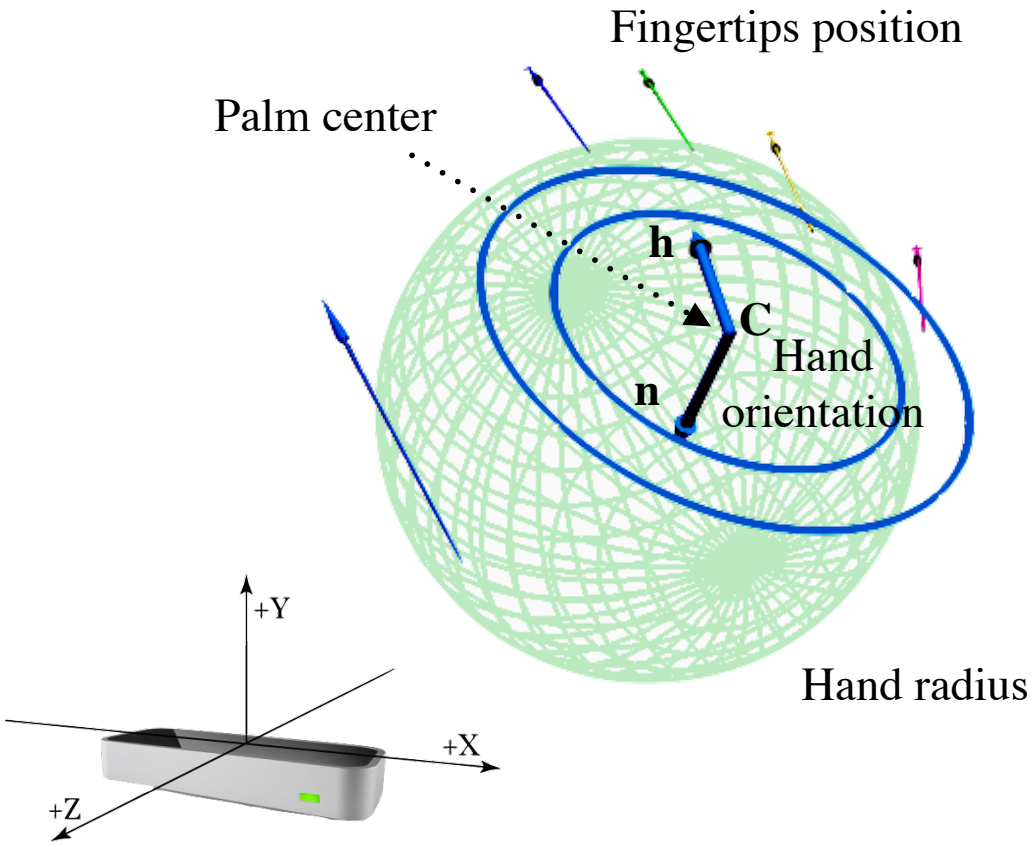


Figure 2.20: Data acquired by the Leap Motion sensor

The 3D positions of the fingertips are quite accurate, compared to the one estimated from the depth data acquired by the Kinect or other similar devices, but their detection is not too reliable. There are some situations, in fact, where the sensor is not able to recognize all the fingers: fingers folded over the palm or hidden from the sensor viewpoint are not captured, and fingers touching each other are sometimes detected as a single finger. Even in situations where the fingers are visible and separated from the hand and the other fingers it may happen that some fingers are lost, specially if the hand is not perpendicular to the camera. Another typical issue of this sensor is that protruding objects near the hand, like bracelets or sleeve edges, can be confused with fingers. These issues are quite critical and must be taken into account in developing a reliable gesture recognition approach, since in different executions of the same gesture the number of captured fingers could vary. For this reason, simple gesture recognition schemes based on the number of the detected fingers or the direct usage of fingertip positions report poor performance.

2.9 Hybrid setup

Leap Motion is raising an high interest in the computer vision research field not only for its affordable cost, its small dimensions and the gesture-based applications it allows to develop quickly, but also because it returns relevant information on the hand pose that could only be obtained in the past with a complex processing of the color and depth data from the framed scene. Most of the computation of the state-of-art gesture recognition approaches of Section 1.2, in fact, is referred to the hand detection, segmentation and extraction of key points that the Leap Motion APIs are able to perform natively in a neglectable time.

For this reason, part of the dissertation is dedicated to the joint usage of the Leap Motion with an affordable depth camera or stereo setup in order to make more efficient and robust the state-of-art automatic hand gesture recognition approaches of Section 1.2. An example of hybrid setup made by the Leap Motion and a Microsoft Kinect (ver.1) is shown in Fig. 2.21, and is used in the experiments described in Chapter 7.

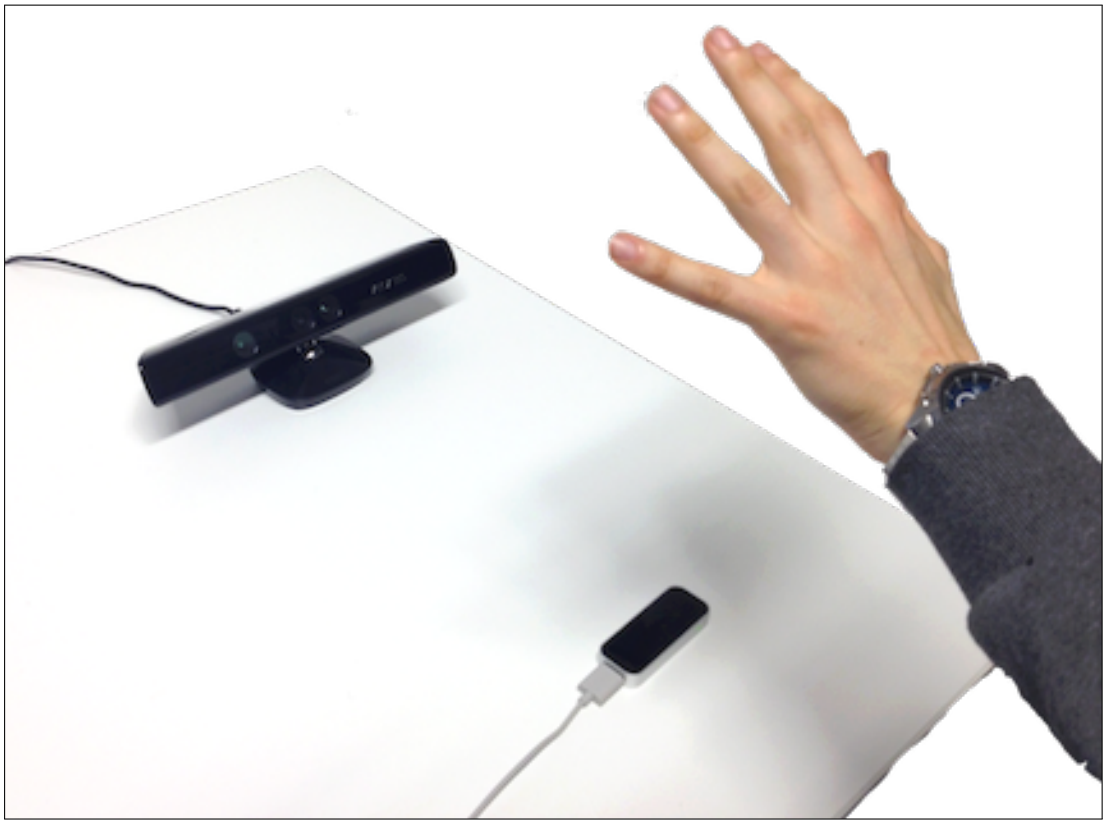


Figure 2.21: Example of hybrid setup made by a Kinect and the Leap Motion

Chapter 3

Hand detection

The first step of the considered gesture recognition pipeline of Fig. 1.8 consists in segmenting the hand from the rest of the scene, since all the information of the performed gesture is entirely encoded in the hand region and in the hand movements. The arm region is, instead, usually discarded as it does not contain any helpful information and its shape and size are affected by the presence of sleeves and bracelets. Hand detection is a crucial step because all the processing in the following chapters is performed on the hand samples only.

Considering one of the acquisition setups described in Chapter 2, the only available data so far are the depth map of the framed scene and, optionally, the related color image. Data from Leap Motion may also be used in this step. The color camera only setup is not considered due to its intrinsic limits.

It is important to recall that, in order to correctly associate the points in 3D space with their projections in the sensor image plane and to perform reliable metric measurements on the scene geometry from the acquired samples, an accurate calibration is mandatory. ToF cameras calibration can be performed by using a checkerboard with known checkers size and the Camera Calibration Toolbox for Matlab [46] or the openCV library [47], since these sensors equip similar optics to the color camera ones and are affected as well by distortion. Note how in this case the color images for calibration are usually replaced by the range camera intensity or infrared maps. A more complete treatment of tof cameras calibration can be found in [1]. In case of Microsoft Kinect (ver.1), a more appropriate calibration protocol is described in [42].

Moreover, in order to take advantage of both color and depth information from the framed scene, a joint calibration of the color and the depth camera is required. Joint calibration, in fact, allows to associate a color and a depth value to each point in the framed scene.

Even though depth information alone may be enough for hand detection purposes when the assumption of the hand being the closest object to camera is valid, a common case for human-machine interfaces empowered by gestures, the proposed framework can exploit both depth and color information or depth and the Leap Motion data in order to recognize the hand more reliably. The implemented detection algorithm either uses depth information only, or also exploits the color or the Leap Motion data according to their availability and the particular application based on gesture recognition.

3.1 Hand detection on depth information only

In applications where the hand is proven to be always the closest object to the sensor, the usage of color information in this phase may be skipped in order to simplify the hand detection procedure and improve the computational performances. The proposed framework in this case follows the pipeline of Fig. 3.1.

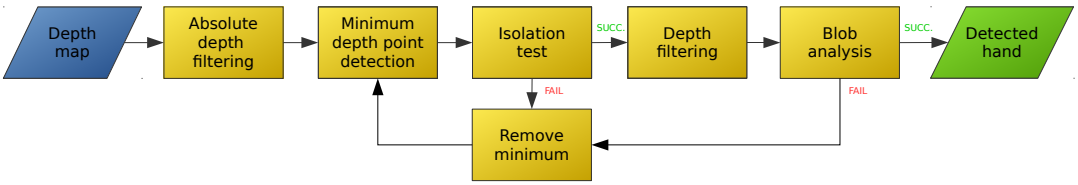


Figure 3.1: Hand detection on depth information pipeline

Assume the only available data is the depth map $D = \{d_{u,v}\} \in \mathbb{R}^{M \times N}$ of the framed scene defined on a lattice Λ_D on the sensor image plane with M rows and N columns. An example of acquired depth map is shown in Fig. 3.2(b).

A first preliminary step in the hand detection pipeline of Fig. 3.1 consists in removing all the possible depth samples with invalid values (e.g., openNI assigns 0 to invalid depth measures for Microsoft Kinect (ver.1)). Moreover, since in several applications employing natural interfaces the user is supposed to interact within a limited volume space in front of a static acquisition setup, a further preliminary step consists in removing all the depth samples $d_{u,v}$ having a relative depth to the acquisition setup higher than a preset threshold T_S (e.g., the work of [21] usually assigns to T_S a value within the range $[1, 1.5]$ meters). Note how such filtering also improves the hand detection performance since most of the background samples are safely removed and no longer considered for further processing.

(a) Acquired color image (b) Acquired depth map (c) Filtered depth map

Figure 3.2: Example of static background removal

Let $D_S = \{d^S_{u,v}\} \in \mathbb{R}^{M \times N}$ denote the depth map D after the preliminary filtering exemplified in Fig. 3.2 and formalized in Eq. 3.1.

$$d^S_{u,v} = \begin{cases} d_{u,v} & \text{if } d_{u,v} \leq T_S \\ \phi & \text{otherwise} \end{cases} \tag{3.1}$$

where ϕ in this case denotes the *null* value for the depth sample in position (u, v) on the sensor lattice to distinguish an invalid measure from a valid depth value 0.

The next step in the hand detection pipeline of Fig. 3.1 is the search for the sample $d^S_{u,v}$ with the minimum depth value D^{min}_S on D_S, with coordinates $\mathbf{d^{S,min}_{u,v}}$, which is likely to be located on one of the fingertips. In order to avoid to select as the closest point an isolated artifact due to measurement noise, the method verifies the presence of an adequate number of depth samples in the neighborhood of the closest point having a similar depth value (e.g., the experiments of Chapter 7 used a rectangular sliding window of 5×5 pixels centered on the $d^{S,min}_{u,v}$ candidate). If the neighborhood of $d^{S,min}_{u,v}$ has an insufficient number of depth samples whose depth value differs not more than a threshold T_W from D^{min}_S, $d^{S,min}_{u,v}$ is discarded by setting $d^{S,min}_{u,v} = \phi$ and a new minimum is searched. The research is performed until a valid minimum is found or D_S has no valid minimums. In this rare case the frame is discarded and the detection is restarted in the next valid frame.

Let now $X_{u,v}$ denote a generic 3D point acquired by the selected range camera, computed as the *back-projection* of the depth value of $d^S_{u,v}$ according to Eq. 3.2.

$$\mathbf{X_{u,v}} = d^S_{u,v} K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.2}$$

with K^{-1} the inverse of the intrinsic parameters matrix of Eq. 2.1 obtained from the sensor calibration. In particular, $X_{u,v}^{min}$ denotes the back-projection of $d_{u,v}^{S,min}$ and is chosen as the starting point for the hand detection procedure.

Once the closest point $X_{u,v}^{min}$ is found, the set of all the points with respectively relative depth and distance from $X_{u,v}^{min}$ lower than two thresholds T_R and T_D is computed by Eq. 3.3.

$$\mathcal{H} = \{X_{u,v} | (d_{u,v}^{S} \leq D_{S}^{min} + T_R) \wedge (\|\mathbf{X_{u,v}} - \mathbf{X_{u,v}^{min}}\| \leq T_D)\} \tag{3.3}$$

The values of T_R and T_D depend on the hand size (typical values are $T_R = 10cm$ and $T_D = 30cm$), that may be estimated during the user calibration phase. In particular, the filtering on the point distances is equivalent to center on $X_{u,v}^{min}$ a sphere of radius T_D and removing all the 3D points $X_{u,v}$ not contained in it. Setting T_R and T_D is a delicate phase that may compromise the further processing: an excessively low value of T_R may discard actual hand samples when the hand is almost perpendicular to the range camera image plane, while an excessively high value may, instead, force the inclusion also of the wrist and the first part of the forearm in $\mathcal{H}$. Analogously, an excessively low value of T_D may discard actual hand samples while an high value is likely to include in $\mathcal{H}$ background samples in the hand neighborhood as well.

It is worth noting that the previous filtering may have retained, due to the measurement noise and to the selected threshold T_R and T_D, background samples constituting artifacts which, if not removed, may seriously compromise all the subsequent steps of the recognition pipeline of Fig. 1.8.

For this purpose, let $B_H = b_{u,v}^{H} \in \{0,1\}^{M \times N}$ be a bidimensional binary mask built on the same lattice Λ_D of D according to Eq. 3.4.

$$b_{u,v}^{H} = \begin{cases} 1 & \text{if } X_{u,v} \in \mathcal{H} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Namely, the entries of B_H are non-zero for the pixel positions corresponding to the points in $\mathcal{H}$.

Starting from B_H, the proposed method applies *blob analysis* techniques to isolate the biggest blob, assumed it is associated to the hand, from the possible smaller ones associated to the retained artifacts. All the points $X_{u,v} \in \mathcal{H}$ associated to the smaller blobs are removed from $\mathcal{H}$, which now only contains 3D points belonging to the hand and part of the forearm. B_H is updated accordingly, setting all the pixels referred to the smaller blobs to 0.

Let now $D_H = \{d^H_{u,v}\} \in \mathbb{R}^{M \times N}$ denote the depth map D_S after the previous filtering, namely the depth map only containing valid depth values for the samples in $\mathcal{H}$. D_H is simply obtained by masking the depth values $d^S_{u,v}$ according to the logic states of the B_H entries (Eq. 3.5).

$$d^H_{u,v} = \begin{cases} d^S_{u,v} & \text{if } b^H_{u,v} = 1 \\ \phi & \text{otherwise} \end{cases} \tag{3.5}$$

A further check is then performed on $\mathcal{H}$ in order to ensure it corresponds to the hand, consisting in measuring the maximum Euclidean distance L^{max}_D between a generic pair of samples in $\mathcal{H}$ (Eq. 3.6) and discarding again the selected minimum $X^{min}_{u,v}$ if L^{max}_D is lower than a preset threshold T_L (e.g., $T_L = 5$cm). This check avoids the selection of $\mathcal{H}$ from an object smaller than any possible hand or from an isolated artifact. If the test fails, a new search for a valid $X^{min}_{u,v}$ has to be performed with a consequent definition of a new candidate hand point cloud $\mathcal{H}$.

$$L^{max}_D = \max_{X^i_{u,v}, X^j_{u,v} \in \mathcal{H}} \|\mathbf{X^i_{u,v}} - \mathbf{X^j_{u,v}}\| \tag{3.6}$$

Finally, note how the lack of other information but depth sometimes leads the detection algorithm to wrong assumptions. For example, when the elbow is nearer than the hand to the acquisition setup as in Fig. 3.3, $X^{min}_{u,v}$ is not located on the fingertips and the actual hand samples are erroneously not included in $\mathcal{H}$.

(a) Acquired color image (b) Acquired depth map (c) Detected hand mask

Figure 3.3: Example of wrong detection on depth map

Fig. 3.4 shows, instead, the intermediate results of the hand detection algorithm for a successful case.

(a) Acquired color image (b) Acquired depth map (c) Detected hand mask

Figure 3.4: Example of correct hand detection on a depth map

3.2 Hand detection on joint color and depth

Hand detection based on depth information only of Section 3.1 is rather effective when the assumption of the hand being the nearest object to the acquisition setup is always valid. Whenever this assumption is no longer verified or the application requires to relax this constraint, integrating the acquired depth data with further information is mandatory for the success of this task.

Consider an hybrid setup made by a color camera and a depth sensor, or a single device providing both color and depth information like Microsoft Kinect 1 or 2, Asus XTION or Creative Senz3D described in Chapter 2. The proposed frameworks implements the algorithms of Fig. 3.5 and 3.6 for the hand detection task exploiting both color and depth information.

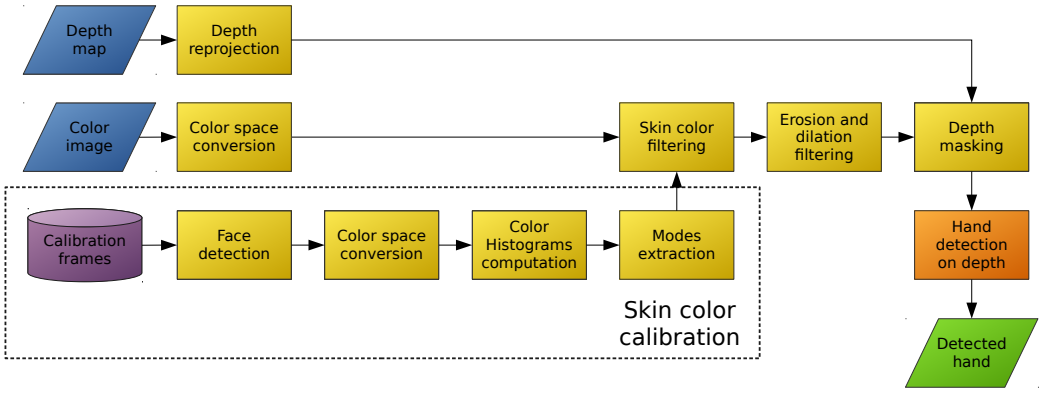


Figure 3.5: Hand detection on joint color and depth data (static skin-color thresholding)

Note how the two pipelines only differ for the usage of static or dynamic skin-color thresholds.

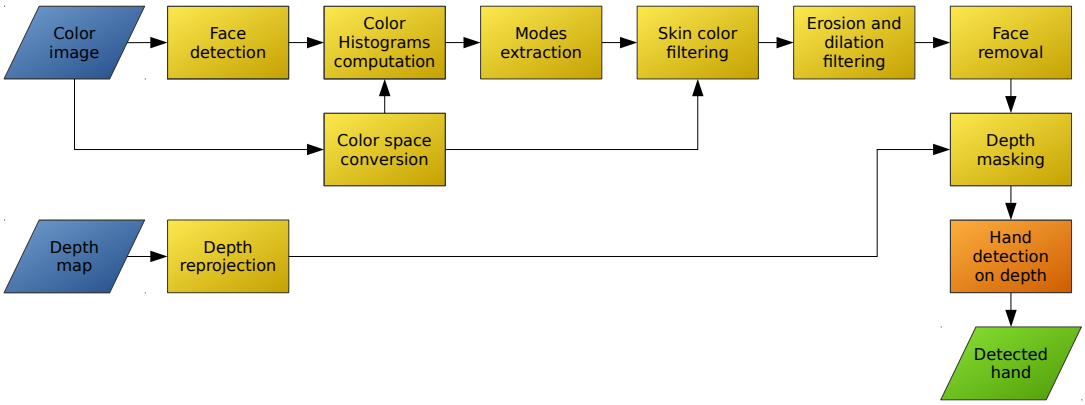


Figure 3.6: Hand detection on joint color and depth data (dynamic skin-color thresholding)

A preliminary yet crucial step in the detection pipelines of Fig. 3.5 and 3.6 consists in reliably associating both color and depth information to each scene sample. Note how, as the only available data are a color image of the framed scene and the related depth map acquired by different sensors, with often different pixel spatial resolutions (usually depth maps have a sensibly lower resolution than color images) there is not a direct correspondence between the color image pixels and the depth map ones.

When a Microsoft Kinect (ver.1) is used, the association can be performed by the tool of [42], while in case of other devices or setups the association has to be performed with ad-hoc solutions. Certain device APIs or middlewares like openNI often align the color image to depth map (or viceversa) automatically, although the achieved accuracy is rather low.

There are mainly three approaches in literature for solving the alignment problem:

- Depth map to color image reprojection
- Point splatting
- Surface rendering

The first method (Fig. 3.7) consists in reprojecting the depth samples $d_{u,v}$ of the acquired depth map D in the color image lattice Λ_C, assigning each depth sample the color of the pixel $c_{u,v}$ of the color image C it "falls" into. Recall that D and C are generally acquired by two different imaging sensors with different lattices Λ_D and Λ_C, where C has often an higher spatial resolution than D. Let

$\mathbf{d_{u,v}} = [u_D \quad v_D]^T \in \mathbb{N}^2$ with $\mathbb{N}$ the natural numbers set denote the coordinates of a generic pixel $d_{u,v}$ of the depth map D, and by $\mathbf{c^D_{u,v}} = [u_C \quad v_C]^T \in \mathbb{R}^2$ the coordinates of the reprojected depth sample $d_{u,v}$ on C according to Eq. 3.7.

$$\widetilde{\mathbf{c}}^{\mathbf{D}}_{\mathbf{u,v}} = K_C(R\mathbf{P_D} + \mathbf{t}) = K_C(RK_D^{-1}\widetilde{\mathbf{d}}_{\mathbf{u,v}} + \mathbf{t}) \tag{3.7}$$

with $\widetilde{\mathbf{c}}^{\mathbf{D}}_{\mathbf{u,v}} = [u_C \quad v_C \quad 1]^T$ the homogeneous coordinates of the reprojected pixel $c^D_{u,v}$, $\widetilde{\mathbf{d}}_{\mathbf{u,v}} = [u_D \quad v_D \quad 1]^T$ the homogeneous coordinates of $d_{u,v}$, K_C and K_D the intrinsic parameter matrices of the color and range cameras respectively, and $(R, \mathbf{t})$ their roto-translation.

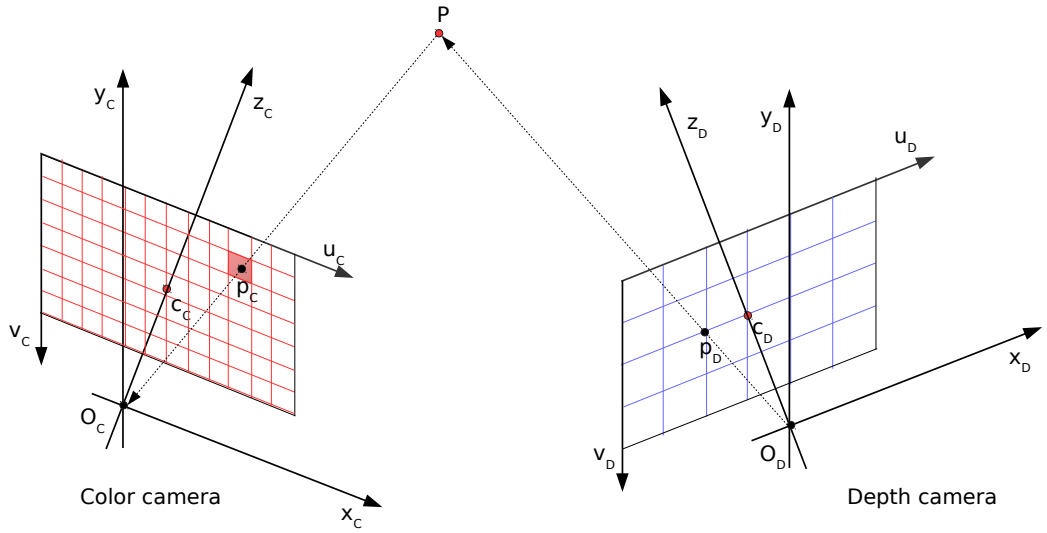


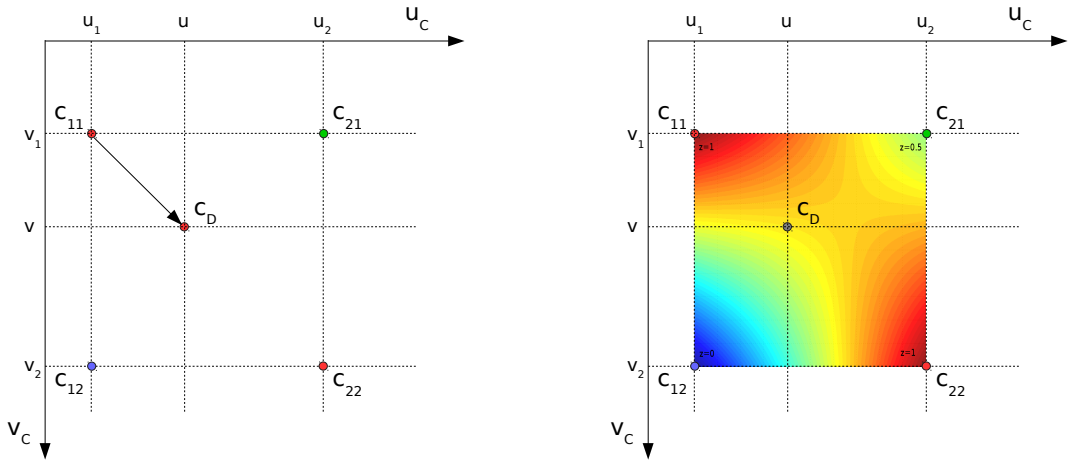
Figure 3.7: Point reprojection

It is worth noting that, due to the different lattices Λ_D and Λ_C, $c^D_{u,v}$ has generally non integer coordinates, thus its value has to be computed by evaluating the four pixels c_{11}, c_{12}, c_{21}, c_{22} in its neighborhood. The simplest method of computing the value of $c^D_{u,v}$ (Fig. 3.8(a)) consists in assigning to $c^D_{u,v}$ the color of the *nearest neighboring* pixel, that is the color of the pixel in $c^D_{u,v}$ neighborhood whose euclidean distance from $c^D_{u,v}$ is minimum (Eq. 3.8).

$$c^D_{u,v} = \underset{c_{u,v} \in \{c_{11}, c_{12}, c_{21}, c_{22}\}}{\operatorname{argmin}} \|\mathbf{c_{u,v}} - \mathbf{c^D_{u,v}}\| \tag{3.8}$$

A more refined approach (Fig. 3.8(b)) computes the value of $c^D_{u,v}$ as the *bilinear interpolation* of the values of the four neighboring pixels (Eq. 3.9).

$$c_{D_{u,v}} = \frac{\begin{array}{c} c_{11}(u_2-u)(v_2-v) + c_{21}(u-u_1)(v_2-v) + \\ c_{12}(u_2-u)(v-v_1) + c_{22}(u-u_1)(v-v_1) \end{array}}{(u_2-u_1)(v_2-v_1)} \tag{3.9}$$



(a) Nearest neighboring pixel color assignment

(b) Bilinear interpolation of the neighboring pixels

Figure 3.8: Comparison of two color computation approaches for the reprojected depth sample on the color image

Finally, it is worth noting that the reprojection method may lead to a color information loss as the only depth samples of D have matched color and depth information.

The second approach (Fig. 3.9), at the basis of the method of [42], allows to associate to each sample $c_{u,v}$ of the color image C a depth sample $d_{u,v}$ of the depth map D: starting from the sparse point cloud of 3D points P_C with coordinates $\mathbf{P_C} = R\mathbf{P_D} + \mathbf{t}$ from Eq. 3.7, expressed in the color camera reference system, the related depth values $d^C_{u,v}$ are computed by *splatting* the points P_C on C and rendering each splat as a Gaussian disk [48]. In case of splat overlapping, the interested pixels $c_{u,v}$ are assigned the lowest depth value among the overlapping splats.

The last method (Fig. 3.10) is the most computational demanding and generally offers better results. It consists in constructing a triangular mesh from the point cloud of the previous approach by triangulating the sparse points and then by extracting the z-buffer from the rendered surface in the color camera view point.

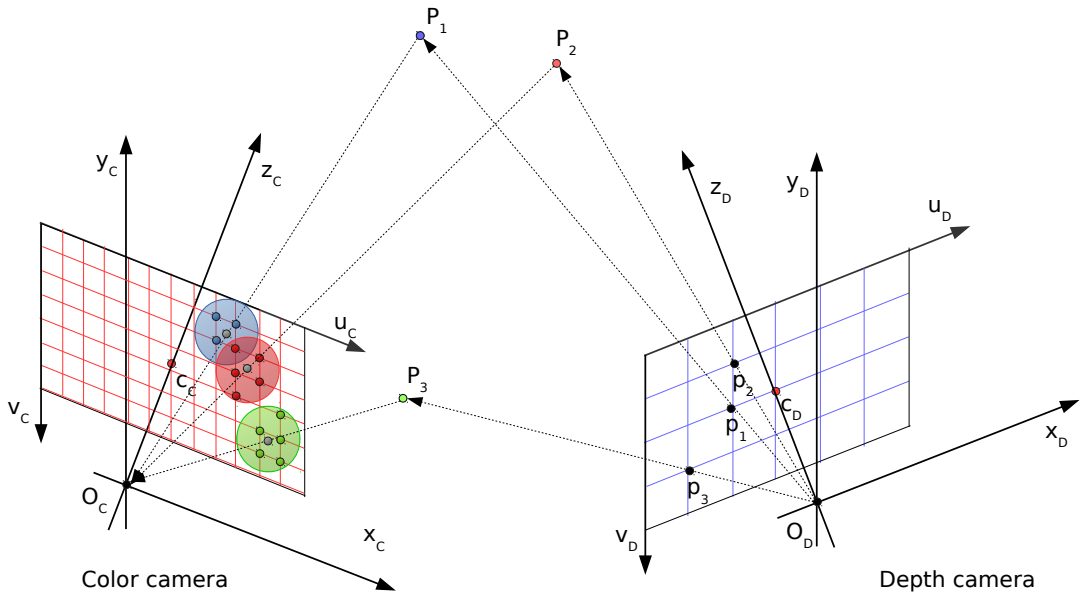


Figure 3.9: Point splatting

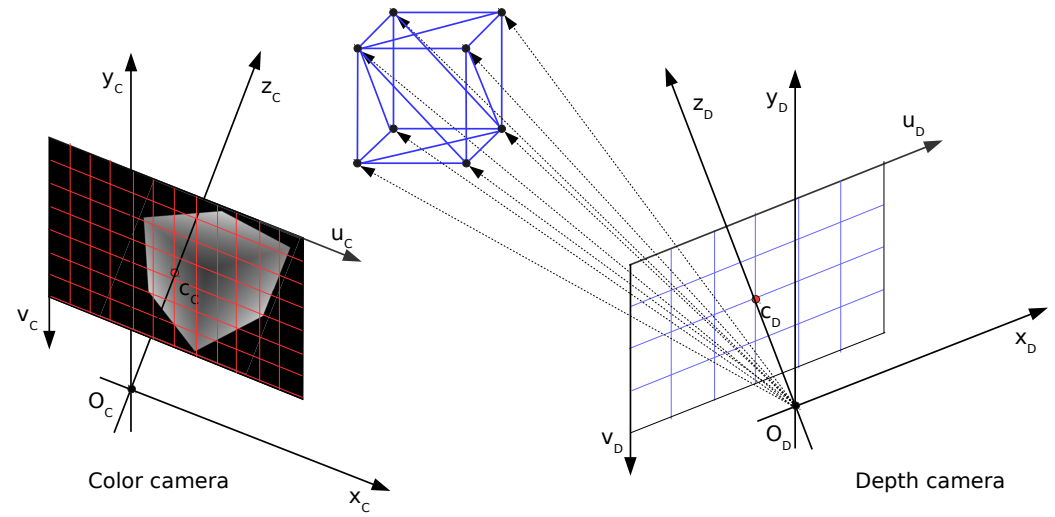


Figure 3.10: Triangulated depth point cloud rendering

Once both color and depth information are associated to each acquired sample, the pipelines in Fig. 3.5 and 3.6 perform a *color thresholding* on C in order to discard all the pixels that are more likely to refer to background samples as their color differs excessively from the reference color of the user's skin. The proposed framework offers two possibilities for executing this task, extending [49]:

- **Static thresholding:** the user's skin color thresholds are determined only in the calibration phase or during the system initialization.
- **Dynamic thresholding:** the user's skin color thresholds are determined adaptively in each acquired frame.

The first method is faster but strongly relies on the accuracy of the skin color range measured during a calibration phase, not always possible. The second approach, instead, is less sensitive to possible skin tone variations due to varying lighting conditions and does not require a skin color calibration phase, though its computational demand may be excessive for certain applications.

Both methods determines the user's skin color thresholds in a limited region roughly corresponding to the nose area, estimated by Viola-Jones [8] face detector or better, since also the depth information is available, with the more robust face detector of [7]. While for the static thresholding the face is only detected on a few calibration frames, for the dynamic version the face detection has to be performed on each frame.

The first important aspect of the acquired color images is their *pixel format*. Most of the low-cost cameras return color images with pixel values expressed in the RGB color space (or RGBA if also the alpha-channel is available), ideal for visualization though unsuitable for several computer vision tasks like the color thresholding. For this reason, both the user's skin color measurement and the color tresholding are then performed on the color image C converted in a proper color space. The selected color space is CIELAB since, as reported in Fig. 3.11(c), leads to better results. One of the reasons of the poor performance of RGB and other color spaces is the dependence of the colors from the luminance, which in turns depends on the varying lighting conditions of the framed scene, and their dependence from the device that generated them. CIELAB space, instead, is designed to map the color distances in actual perceived color differences and is device independent, but also separates the luminance (channel L) from the color components (a and b).

Let $C_{Lab} = \{c^{Lab}_{u,v}\}$, where $c^{Lab}_{u,v}$ has coordinates $\mathbf{c^{Lab}_{u,v}} = [c_a \quad c_b]^T$, denote the color image C converted in the CIELAB color space. The current step in the

(a) Acquired color image (b) Acquired depth map (c) Thresholding on RGB

(d) Thresholding on RGB (Kovac's rule) (e) Thresholding on HSV (f) Thresholding on Lab

Figure 3.11: Example of skin color thresholding masks on different color spaces

pipelines of Fig. 3.5 and 3.6 consists in building an histogram of the color distribution for each separate channel within the previously detected region. Note how for the pipeline of Fig. 3.6 the histograms are referred to a single frame, while for the pipeline of Fig. 3.5 they are averaged on a few calibration frames in order to obtain a more robust estimate. Note also how the L channel is, indeed, discarded for the further thresholding as it accounts for the unstable luminance.

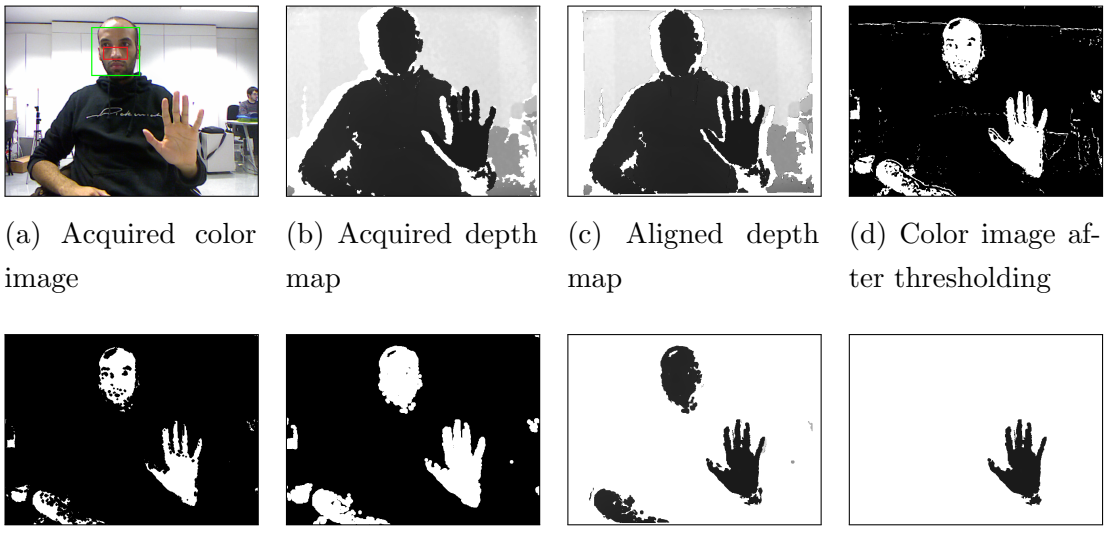
The *modes* M_a and M_b of the color distributions are selected as the base skin color components. The skin color filtering in Fig. 3.5 and 3.6 is a two-fold thresholding consisting in firstly selecting all the pixels of C_{Lab} whose color distance from the reference skin color c_S defined by M_a and M_b is lower than a pair of highly selective thresholds $T_a^{1^{st}}$ and $T_b^{1^{st}}$ denoting the maximum relative component distances of a generic pixel color from c_S, and then by selecting the discarded pixels in the first pass whose color distance is lower than a pair of more relaxed thresholds $T_a^{2^{nd}} > T_a^{1^{st}}$ and $T_b^{2^{nd}} > T_b^{1^{st}}$ if the number of selected pixels in the first pass in their neighborhoods is higher than a given threshold T_ρ. Let $B_C = \{b_{u,v}^C\}$ define a binary masks on the same lattice Λ_C of C indicating what pixels of C_{Lab} are retained by the color thresholding, as formalized in Eq. 3.10.

$$b_{u,v}^{C,1^{st}} = \begin{cases} 1 & \text{if } |c_a - M_a| \leq T_a^{1^{st}} \wedge |c_b - M_b| \leq T_b^{1^{st}} \\ 0 & \text{otherwise} \end{cases}$$
$$b_{u,v}^{C,2^{nd}} = b_{u,v}^{C,1^{st}} \vee \begin{cases} 1 & \text{if } |c_a - M_a| \leq T_a^{2^{nd}} \wedge |c_b - M_b| \leq T_b^{2^{nd}} \\ & \wedge \sum_{(u,v)\in W(c_{u,v}^{Lab})} b_{u,v}^{C,1^{st}} \geq T_\rho \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

where $W(c_{u,v}^{Lab})$ denotes a sliding window centered on pixel $c_{u,v}^{Lab}$.

A further *erosion* followed by a *dilation* filtering on the resulting binary mask B_C first removes the smallest blobs due to the filtering noise and then expands the retained ones, in particular the blobs referred to the hands, in order to include in B_C hand pixels in C_{Lab} that may have been previously discarded by the color filtering. B_C is then applied to D_R, the depth map D aligned with C by reprojection (Eq. 3.7), to only select the regions int the depth map most likely to be referred to the hands. Note how this approach may also be used to detect both hands, as they are masked by two different blobs.

Finally, hand detection on depth data is performed on D_R with the approach of Section 3.1, since D_R may still contain artifacts not removed by the previous processing. An example of hand detection on joint color and depth information is shown in Fig. 3.12.



(a) Acquired color image (b) Acquired depth map (c) Aligned depth map (d) Color image after thresholding

(e) Thresholded image after erosion (f) Eroded image after dilation (g) Masked aligned depth map (h) Detected hand

Figure 3.12: Example of hand detection with joint color and depth information

The joint usage of color and depth information may also be exploited to relax the static background thresholding of Section 3.1 in order to allow the user to freely move within the acquisition setup viewing volume. Since the head position in the 3D space may be reliably estimated with the approach of [7], it is possible to detect the maximum volume occupiable by the user and consequently define dynamic thresholds for the foreground and background sample removal of Eq. 3.1. While this is not usually needed when the user is sitting in front of a computer or a machine, this possibility may become mandatory when the user has to interact with the interface while changing position.

3.3 Hand detection on joint depth and Leap Motion data

Hand detection task is way simpler when the Leap Motion is jointly used with a range camera in the same acquisition setup. Assume, in fact, the setup is calibrated, thus the data provided by the Leap Motion can be expressed in the range camera coordinate system with a simple roto-translation between the two coordinate systems. In particular, let C_D denote the hand center estimated by the Leap Motion software expressed in the range camera coordinate system.

C_D can now replace $X_{u,v}^{min}$ in Eq. 3.3 and the relative distance threshold can T_R be halved, since C_D roughly lies in the hand center and not on a fingertip. Hand detection then continues as in Section 3.1.

It is worth noting that in this case no color information is required to relax the assumption of the hand being the nearest object to the camera and the algorithm does not need to reiterate on different $X_{u,v}^{min}$ candidates as C_D is reliable enough.

Chapter 4

Hand segmentation

Hand detection of Chapter 3 reliably segments the hand from the background. Differently from earlier color only based approaches resumed in Section 1.2, the work in this thesis leverages mostly depth information for this task and, whenever available and necessary, exploits also color cues or the key points provided by the Leap Motion APIs.

However, detection itself only provides an insufficient amount of information for the estimation of the performed gestures. The necessary information is, in fact, contained in the feature sets described in Chapter 5, which in turn require for their extraction some additional cues like the hand orientation, the hand center and the location of the palm and fingers regions.

Hand segmentation in this thesis, following the scheme of Fig. 4.1, relies on the estimation of a local 3D reference system set on the expected hand center and representing the hand orientation. Note how this coordinate system has a fundamental importance for the extraction of several features in Chapter 5.

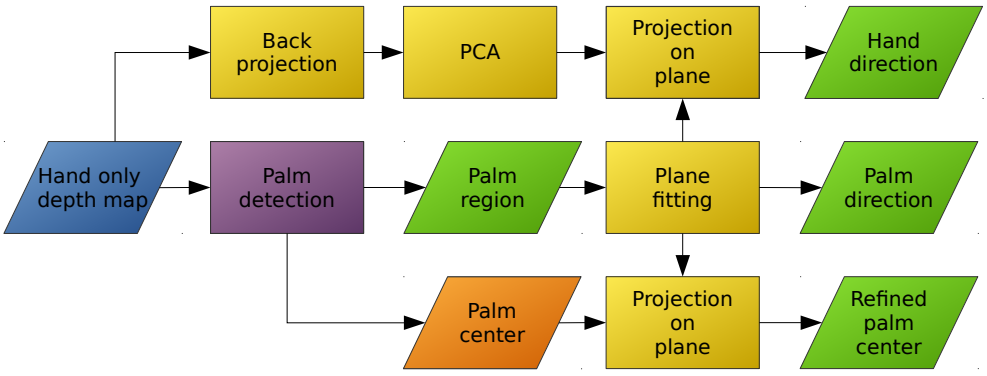


Figure 4.1: Hand segmentation pipeline

4.1 Palm detection

The first step in the segmentation pipeline of Fig. 4.1 is the detection of the palm region within D_H, the acquired depth map D after hand detection in Chapter 3.

Recall that the described detection method returns, beside a depth map $D_H = \{d^H_{u,v}\}$ only containing valid depth measures for the pixel positions referred to the expected hand region, a point cloud $\mathcal{H}$ obtained by back-projecting (Eq. 3.2) the pixels of D_H in the 3D space, and a binary mask B_H reporting the positions of the hand region pixels in D_H.

In particular, the binary mask B_H is at the basis of two different palm detection approaches implemented in the proposed framework:

- Circle fitting
- Ellipse fitting

The first approach consists in fitting the largest inscribed circle $\mathcal{C}$ with center c_p and radius r_p in the expected palm region in the binary mask B_H. The choice of the circle as geometric shape is due to its *rotational invariance*.

The second approach is, instead, an improvement of the first one designed to overcome its limits in dealing with narrow or excessively slanted palms respect to the range camera image plane.

4.1.1 Circle fitting approach

Palm detection based on circle fitting follows the pipeline in Fig. 4.2.

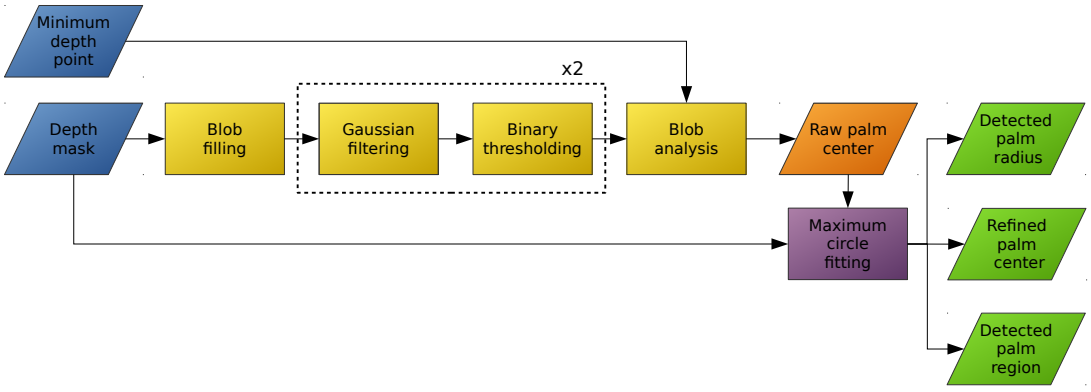


Figure 4.2: Palm detection with circle expansion pipeline

The first crucial step for the circle fitting success is the choice of a good starting point c_0 for the circle expansion. The implemented selection strategy is

based on the fact that the palm region in B_H has the highest point density, since usually the palm area is larger than the fingers and the wrist. Following this rationale, a combination of proper image filtering algorithms and blob analysis is able to detect the highest density blob in B_H containing c_0.

More specifically, B_H is first convoluted with a 2D Gaussian kernel (Eq. 4.1) with a very large standard deviation $\sigma = (\sigma_u, \sigma_v)$, obtaining a strongly blurred grayscale image $I_G = \{i^G_{u,v}\}$ with values proportional to the point density.

$$i^G_{u,v} = \sum_{k,l} b^H_{u+k,v+l} g^\sigma_{k,l} \tag{4.1}$$

where $g^\sigma_{k,l} = \frac{1}{2\pi\sigma_k\sigma_l} \exp\left(-\frac{k^2}{2\sigma_k^2} - \frac{l^2}{2\sigma_l^2}\right)$ denotes the 2D gaussian kernel coefficient at position (k, l).

It is worth noting that, since the hand region area in B_H varies not only according to the type of performed gesture but with the minimum distance of the hand from the acquisition setup as well, a fixed value for σ would lead to different filtering results for each acquired frame. For this reason, σ is scaled according to Eq. 4.2 in order to dynamically adapt to the hand distance from the acquisition setup.

$$\sigma_D = \sigma_0 \frac{1}{D^{min}} \tag{4.2}$$

where σ_0 is the base value for σ (e.g., for the tests of Chapter 7 σ_0 has been set to $^1/_4$ of the width of B_H for both its components) and D^{min} is the shortest distance of the hand points in $\mathcal{H}$ from the acquisition setup, computed in Chapter 3.

Scaling by Eq. 4.2 makes the window size in metric units invariant from hand distance from the acquisition setup, and ensures that the support of the filter is always large enough to capture the thickness of the hand or arm regions.

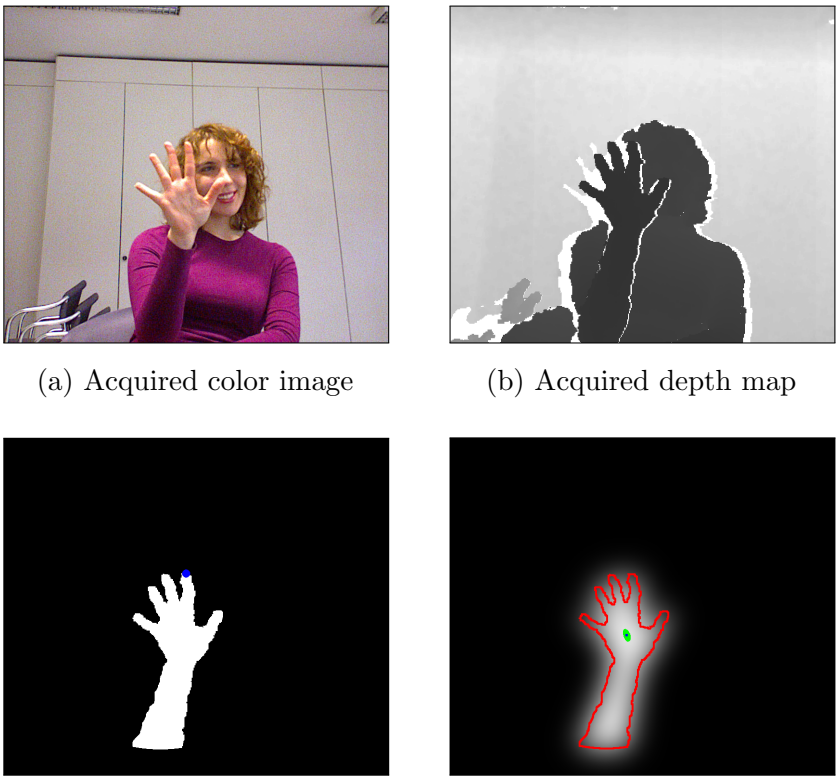
Let $I^{max}_G = \max_{u,v} i^G_{u,v}$ denote the maximum computed density, and $T^\rho_G \in [0, 1]$ a threshold value (in the experiments of Chapter 7 $T^\rho_G = 0.9$, namely $T^\rho_G I^{max}_G$ corresponds to the 90% of the maximum density). A binary thresholding on I_G by T^ρ_G returns a new binary mask $B_G = \{b^G_{u,v}\}$ (Eq. 4.3) made of one or more blobs representing possible candidates to contain c_0. This is also due to the fact that there may be more than one pixel in I_G with value I^{max}_G.

$$b^G_{u,v} = \begin{cases} 1 & \text{if } i^G_{u,v} \geq T^\rho_G I^{max}_G \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

The value of T_G^ρ represents a trade-off between the size and number of blobs in B_G and the estimated c_0 position accuracy: relaxed thresholds are more likely to return more than one large blob probably containing the searched c_0, while tighter thresholds usually return only one blob with a limited area but also with an higher risk of not containing c_0. In some unlucky cases, in fact, c_0 may not lie near the actual palm center, but rather in the arm region if the arm points density is higher than the hand ones.

In order to over reduce this risk, the proposed algorithm prefers a less tight threshold T_G^ρ to retain an higher number of blobs and performs a second-pass of filtering and thresholding on B_G. The idea is eroding the minor blobs to only retain the main one, supposed to contain the desired c_0.

Finally, since B_G after the filtering may still contain more than one blob, a further blob analysis is performed on B_G to compute the *center of mass* for each retained blob, and the nearest center of mass to the projection $d_{u,v}^{min}$ of $X_{u,v}^{min}$ on B_G is chosen as c_0. The rationale is that the actual palm center c_p can not be located too far from the nearest hand point to the acquisition setup.



(a) Acquired color image (b) Acquired depth map

(c) Detected hand mask (d) Computed c_0 (in blue)

Figure 4.3: Example of computed c_0 for circle fitting algorithm initialization

Once a suitable starting point c_0 for the palm detection has been determined, the algorithm formalized in Alg. 4.2 computes iteratively the maximum circle that can be inscribed in the palm region with center c_p in c_0 neighborhood. Alg. 4.2 starts by expanding a circle $\mathcal{C}$ with initial radius $r = r_0$ (with $r_0 = 1$ pxl in the current implementation) and center $c = c_0$ in B_H, using Alg. 4.1, until the density ρ of the mask pixels within $\mathcal{C}$ is higher than a preset threshold T_ρ (e.g., $T_\rho = 0.95$, that is the 95% of the points within $\mathcal{C}$ must be referred to samples of $\mathcal{H}$). The tolerance of T_ρ accounts for errors due to noise or artifacts of the depth sensor.

Algorithm 4.1 Circle expansion algorithm

Input:

$B = \{b_{u,v}\}$: binary mask

$c = (c_u, c_v)$: circle center position c with coordinates c_u, c_v

r_0: initial radius length [pxl]

ρ^{min}: minimum circle point density

s: radius increment step [pxl]

Output:

r_p: maximum inscribed circle radius [pxl]

ρ_f: maximum inscribed circle point density

function MAXEXPAND(c, r_0,ρ^{min},s)

  $r_p \leftarrow r_0 - s$

  repeat

    $r_p \leftarrow r_p + s$,$\rho_f \leftarrow 0$

    $A_B \leftarrow 0$ ▷ Number of hand pixels within the circle

    $A_C \leftarrow 0$ ▷ Number of pixels within the circle

    for all (u, v) s.t. $(u - c_u)^2 + (v - c_u)^2 \leq r_p^2$ **do**

      $A_C \leftarrow A_C + 1$

      if $b_{u,v}$ is *true* **then**

        $A_B \leftarrow A_B + 1$

      end if

    end for

    if $A_B/A_C \leq \rho^{min}$ **then**

      $\rho_f \leftarrow A_B/A_C$

    end if

  until $\rho_f > \rho^{min}$

  return r_p, ρ_f

end function

Algorithm 4.2 Circle fitting algorithm

Input:

$B = \{b_{u,v}\}$: binary mask

$c_0 = (c_u^0, c_v^0)$: initial circle center position c_0 with coordinates c_u^0, c_v^0

T_ρ: minimum circle point density threshold

s: radius increment step [pxl]

Output:

r_p: maximum inscribed circle radius [pxl]

c_f: maximum inscribed circle center position with coordinates c_u, c_v

$c_f \leftarrow c_0$

$(r_p, \rho_f) \leftarrow$MAXEXPAND(c_0,1,ρ^{min},s)

repeat

  $r \leftarrow 1$, $\rho \leftarrow 0$

  $c_n = (c_u, c_v - s)$ ▷ Circle center after up shift

  $r_n, \rho_n \leftarrow$MAXEXPAND(c_n,r_p,T_ρ,s)

  $c_s = (c_u, c_v + s)$ ▷ Circle center after down shift

  $r_s, \rho_s \leftarrow$MAXEXPAND(c_s,r_p,T_ρ,s)

  $c_e = (c_u - s, c_v)$ ▷ Circle center after right shift

  $r_e, \rho_e \leftarrow$MAXEXPAND(c_e,r_p,T_ρ,s)

  $c_w = (c_u + s, c_v)$ ▷ Circle center after left shift

  $r_w, \rho_w \leftarrow$MAXEXPAND(c_w,r_p,T_ρ,s)

  if $r_n > r \vee (r_n = r \wedge \rho_n > \rho)$ **then**

    $r \leftarrow r_n$, $\rho \leftarrow \rho_n$

  end if

  if $r_s > r \vee (r_s = r \wedge \rho_s > \rho)$ **then**

    $r \leftarrow r_s$, $\rho \leftarrow \rho_s$

  end if

  if $r_e > r \vee (r_e = r \wedge \rho_e > \rho)$ **then**

    $r \leftarrow r_e$, $\rho \leftarrow \rho_e$

  end if

  if $r_w > r \vee (r_w = r \wedge \rho_w > \rho)$ **then**

    $r \leftarrow r_w$, $\rho \leftarrow \rho_w$

  end if

  if $r > r_p$ **then**

    $r_p \leftarrow r$

  end if

until $r > r_p$

After the maximum radius value satisfying the threshold is found, the coordinates of c are shifted towards the direction that leads to the maximum expansion of the shifted circle. In case more than one direction leads to the maximum expansion, c is shifted to the direction among them having the maximum point density within the expanded circle.

Then, the two phases keep iterating until the largest possible circle has been fitted on the palm area. The final position of c, denoted by c_p, represents the estimated palm center and will be the starting point for the further processing.

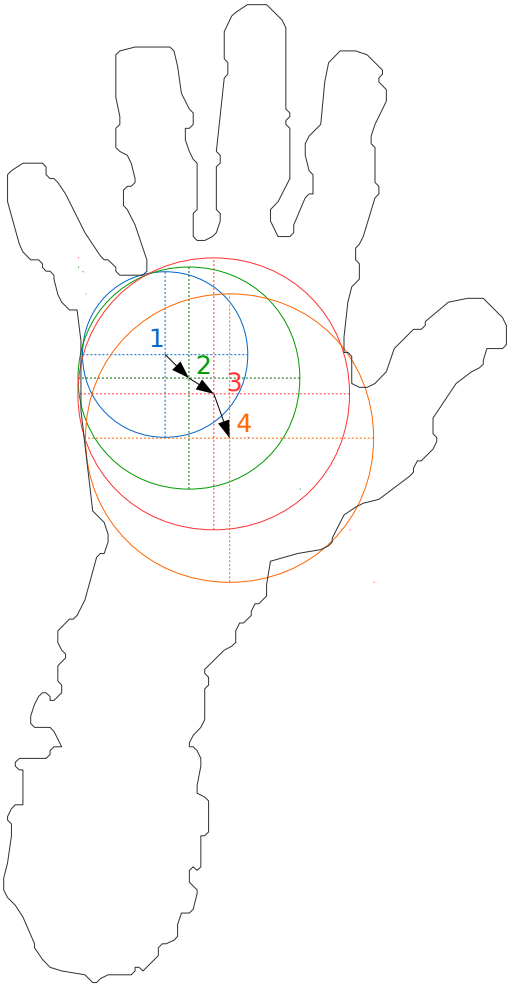


Figure 4.4: Palm detection with circle expansion

The corresponding 3D point C_P obtained by back-projection of c_p, that from now on will be referred to as the *centroid* of the hand, will play an important role in the proposed algorithm together with the final radius value r_p. Moreover, note how the position of the centroid is also useful in order to reconstruct the trajectory followed by the hand in dynamic gestures, necessary in several applications (e.g.,

for the control of virtual mouses or of browsing of 3D scenes) and is one of the key points for the recognition of dynamic gestures.

Alg. 4.2 effectiveness is clearly dependent on the selection of the starting point c_0 for the first circle expansion: if c_0 lies within the palm region, not necessarily near the actual palm center, the algorithm will converge quickly to an optimum of the estimated palm center position. Conversely, if the starting point lies in one of the fingers the circle expansion will probably stop early leading to wrong estimations (e.g., palm center confined in a phalanx).

4.1.2 Ellipse fitting approach

Palm detection by circle fitting of Section 4.1.1 allows to obtain a reasonable but not always accurate estimate of the palm region in the depth mask B_H. This happens for two main reasons:

1. the palm may be sensibly longer than wide, e.g., for people having thin hands.

2. In several acquired gestures the hand is not parallel to the imaging plane and the palm shape is then distorted by the projection of the hand on the range camera image plane.

In order to deal with these issues, Fig. 4.5 proposes a refined hand detection algorithm based on computing the ellipse best approximating the expected palm region boundary [50].

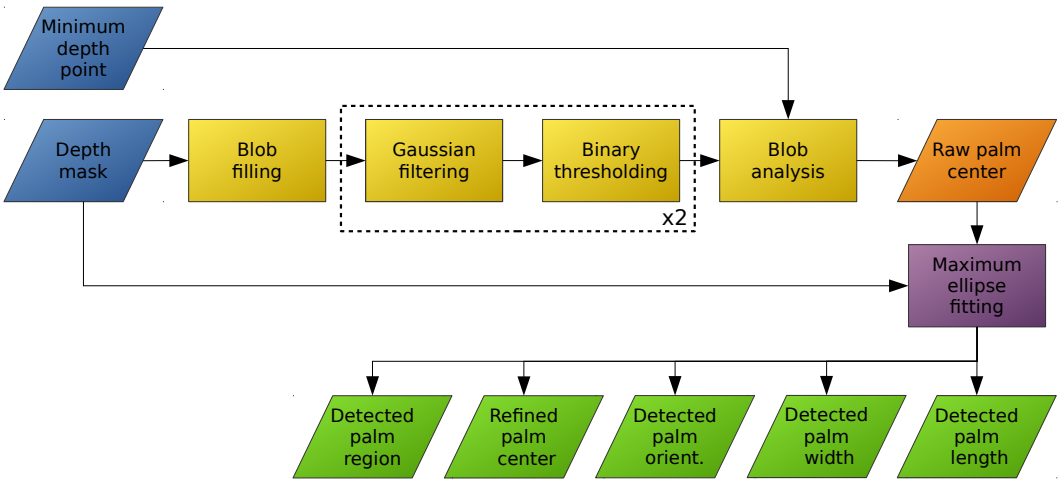


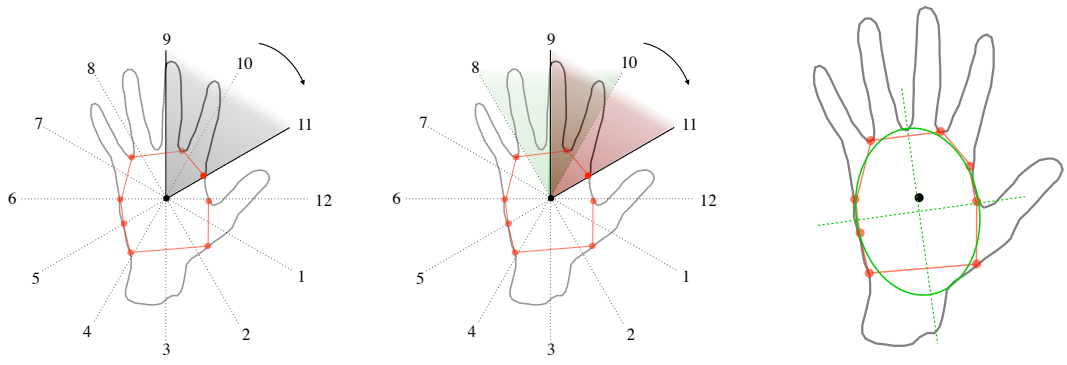
Figure 4.5: Palm detection with ellipse fitting pipeline

Consider again B_H, the binary mask only containing a single blob representing the hand sample positions in the related depth map D_H. Let $\mathcal{B}_H^e = \partial B_H$ denote the hand contour point set obtained from *edge detection* on B_H (e.g, with Canny method [51]). The palm region boundary is detected by intersecting N overlapping angular sectors S_i for $i = 1, 2, \ldots, N$ with $\mathcal{B}_H^e$ (Fig. 4.6), returning each one a subset $\mathcal{S}_i \subseteq \mathcal{B}_H^e$ of the contour points coordinates of the points contained in Sector S_i. Each sector S_i contributes for a single palm contour point $p_i \in \mathcal{S}_i$ computed as the *nearest* point of $\mathcal{S}_i$ to the approximated palm center c_p (Eq. 4.4).

$$p_i = \arg\min_{p_j \in \mathcal{S}_i} \|\mathbf{p_j} - \mathbf{c_P}\| \tag{4.4}$$

Note how c_0 can be also be used instead of c_P to speed up the ellipse computation, avoiding the circle fitting, although when c_0 is positioned near the actual palm boundary the latter is not partitioned uniformly by the sectors S_i thus returning a less reliable sample of the palm contour.

Moreover, while an higher value of N leads to a more accurate palm region detection, this also reduces the sectors area, with an higher risk of extracting edge points from the fingers region than from the palm one.



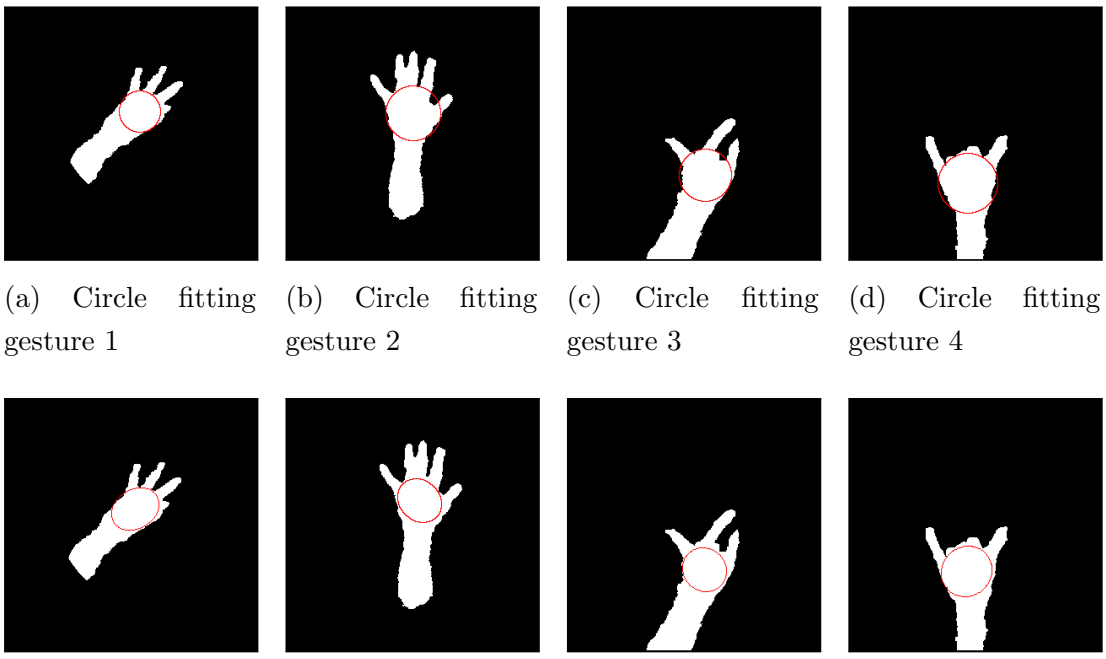
(a) First pass (b) Second pass (c) Fitted ellipse

Figure 4.6: Ellipse fitting for palm detection

The extracted points correspond to the corners of a polygon contained inside the hand contour $\mathcal{B}_H^e$ and that approximates the hand palm, as depicted in Fig. 4.6. The choice of using partially superimposed sectors and to take the minimum distance inside each sector ensures that the polygon corners are chosen at the basis of the fingers and that the finger samples are not included in the polygon.

Once the palm region approximating polygon has been determined, the palm detector exploit the method of [52] to find the ellipse that better approximates the polygon in the least-square sense.

Finally, Fig. 4.7 compares the palm detection results of the circle and ellipse fitting approaches on a few gestures.



(a) Circle fitting gesture 1 (b) Circle fitting gesture 2 (c) Circle fitting gesture 3 (d) Circle fitting gesture 4

(e) Ellipse fitting gesture 1 (f) Ellipse fitting gesture 2 (g) Ellipse fitting gesture 3 (h) Ellipse fitting gesture 4

Figure 4.7: Comparison between circle and ellipse fitting algorithms for palm detection

4.2 Hand orientation estimation

The second main step in the segmentation pipeline of Fig. 4.1 is the estimation of the hand orientation respect to the acquisition setup, defined by two components:

- **hand main direction:** the main direction in 3D space of the fingers.
- **Palm direction:** the direction the palm is pointed to, orthogonal to the fingers one.

The third direction (axis) of the hand coordinate system will be computed as the orthogonal axis to the hand and palm directions according to the *right-hand rule*.

4.2.1 Palm orientation estimation

Palm direction is computed as the *normal* of a 3D plane π fitted on the point cloud of the palm samples, following the rationale the actual palm samples lie on a ratherly flat surface in 3D space.

Let $\mathcal{P} \subset \mathcal{H}$ denote the subset of $\mathcal{H}$ corresponding to the palm samples, that can be easily computed by Eq. 4.5.

$$\mathcal{P} = \{X_{u,v} \in \mathcal{H} | (u - c_u^p)^2 + (v - c_v^p)^2 \leq {r_p}^2\} \tag{4.5}$$

where $\mathbf{c_p} = [c_u^p \quad c_v^p]^T$ denotes the coordinates of the center of the circle best fitting the palm region and r_p its radius. Note how in case of ellipse fitting Eq. 4.5 is replaced by Eq. 4.6.

$$\mathcal{P} = \left\{ X_{u,v} \in \mathcal{H} | R_E \left[\left(\frac{u-c_u^p}{a} \right)^2 \quad \left(\frac{v-c_v^p}{b} \right)^2 \right]^T \leq 1 \right\} \tag{4.6}$$

where a and b are the semi-axis lengths and R_E the rotation matrix representing the ellipse orientation respect to the image plane coordinate system.

The classic plane fitting approach on $\mathcal{P}$ is based on the *orthogonal distance regression* which in turn exploits the *single value decomposition* (SVD) for estimating the plane parameters minimizing the square sum of the orthogonal distances between the palm points and the estimated plane.

Let π denote a generic plane defined by two parameters: c, a generic point lying on π and $\mathbf{n} = [n_x \quad n_y \quad n_z]^T$ the plane normal. Let also $p_i \in \mathcal{P}$ denote a generic 3D point of the palm point cloud containing N points. c can be found by solving Eq. 4.7:

$$(c, \mathbf{n}) = \underset{\mathbf{c}, \|n\|=1}{\operatorname{argmin}} \sum_{i=1}^{N} ([\mathbf{p_i} - c]^T \mathbf{n})^2 \tag{4.7}$$

which it is provable to result in $c = \overline{p}_i = \frac{1}{N} \sum_{i=1}^{N} p_i$, namely c is the *center of mass* of the 3D points p_i in $\mathcal{P}$. The idea of Eq. 4.7 is that if a point p_i actually lies on plane π, then by definition the vector $(\mathbf{p_i} - \mathbf{c})$ must be orthogonal to the plane normal $\mathbf{n}$. The best plane π according to Eq. 4.7 is, thus, the one that minimizes the average "orthogonality error" defined as the distance from the orthogonality condition due for a improper estimation of the plane parameters.

Now, by defining $A \triangleq [p_1 - c \quad p_2 - c \quad \dots \quad p_N - c] \in \mathbb{R}^{3\times N}$, the problem of Eq. 4.7 may be reformulated as:

$$\mathbf{n} = \underset{\|n\|=1}{\operatorname{argmin}} \|A^T\mathbf{n}\|_2^2 \tag{4.8}$$

Using the singular value decomposition $A = U\Sigma V^T$ with $U \in \mathbb{R}^{3\times3}$ and $V \in \mathbb{R}^{N\times N}$ orthogonal matrices, and $\Sigma \in \mathbb{R}^{3\times N}$ diagonal matrix with diagonal entries $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq 0$ called *singular values*. It follows that $\|A^T\mathbf{n}\|_2^2 = \|V\Sigma^T U^T\mathbf{n}\|_2^2 = \|\Sigma^T U^T\mathbf{n}\|_2^2 = (\sigma_1 y_1)^2 + (\sigma_2 y_2)^2 + (\sigma_3 y_3)^2$ where $\mathbf{y} = U^T\mathbf{n}$ is a unit vector. Thus, $\|A^T\mathbf{n}\|_2^2$ is minimized by $\mathbf{y} = [0 \quad 0 \quad 1]^T$ or equivalently $\mathbf{n} = \mathbf{u_3}$ with $\mathbf{u_3}$ 3^{rd} column of U. Moreover, the plane fitting error, that is the minimum of the sum of squared distances $\|A^T\mathbf{n}\|_2^2$ for the estimated plane, is simply σ_3^2.

It is worth noting that, while the previous plane fitting approach is theoretically correct, in practical situations it is likely to often fail due to the noise in the acquired depth map. For this reason, the proposed framework encloses the the plane fitting with SVD in a more robust plane fitting algorithm based on RANSAC [53], as described in Alg. 4.3.

This enforcement ensures the eventual noisy samples will not lead detrimental effects to the plane estimation, as they are considered outliers in the plane model. Ransac requires a proper setting of the value of the outlier threshold T_O that, in the plane model, corresponds to the maximum distance from the plane a candidate point must have to be considered an inlier. Such distance should not be lower than the acquisition system accuracy, as the sample noise could lead Alg. 4.3 to discard several samples for the model estimation.

The plane normal $\mathbf{n}$ returned by Alg. 4.3 from now on will be denoted as $\mathbf{z_p}$, referring to the third axis of the local hand coordinate system.

Note how the estimated $\mathbf{z_p}$ direction may, sometimes, point to the hand dorsum instead to the acquisition setup. This ambiguity is easily solvable by inverting the axis direction if the angle formed by $\mathbf{z_p}$ and the optical axis $\mathbf{z}$ is acute, whenever the palm in the gestures of the employed dictionary is supposed to always face the acquisition setup. When, instead, the dictionary also accounts for gestures showing either the palm or the hand dorsum, other information (e.g., tracking from the previous frames) is required to solve this ambiguity.

Finally, it is worth noting that $\mathcal{P}$ obtained from Eq. 4.5 or Eq. 4.6 may, indeed, also contain 3D points belonging to the possible folded fingers, as the palm detection approaches of Sections 4.1.1 and 4.1.2 are based on the hand binary mask only. For this reason, all the 3D points $X_i \in \mathcal{P}$ whose signed distance

from the palm plane π is higher than a given threshold T_π (e.g., $T_\pi = 20mm$) have to be removed from $\mathcal{P}$ as they are likely to belong the the fingers. After the plane fitting, then, the actual palm points can be extracted from $\mathcal{P}$ with Eq. 4.9.

$$\mathcal{P} = \mathcal{P} \setminus \{X_i \in \mathcal{P} | [\mathbf{X_i} - \mathbf{C_P}]^T \mathbf{z_p} < 0 \vee [\mathbf{X_i} - \mathbf{C_P}]^T \mathbf{z_p} > T_\pi\} \tag{4.9}$$

Algorithm 4.3 Ransac plane fitting

Input:

$\mathcal{P}$: palm point cloud

$n \leftarrow 3$: the minimum number of data values required to fit the plane

k: the maximum number of iterations allowed in the algorithm

T_O: a threshold value for determining when a data point fits a model

d: the number of samples required to assert that a model fits well to data

Output:

$(c, \mathbf{n})$: estimated plane reference point in 3D space and normal

$it \leftarrow 0$, $(c^{best}, \mathbf{n^{best}}) \leftarrow \phi$, $besterr \leftarrow \infty$

while $it < k$ **do**

  $\mathcal{P}_3 \leftarrow$ 3 randomly selected points from $\mathcal{P}$

  $(c, \mathbf{n}, err) \leftarrow$ FITPLANESVD($\mathcal{P}_3$)

  $\mathcal{I} \leftarrow \phi$

  for all $p \in \mathcal{P} \setminus \mathcal{P}_3$ **do**

    if DISTANCEFROMPLANE(p,c,$\mathbf{n}$) $< T_O$ **then**

      $\mathcal{I} \leftarrow \mathcal{I} \cup \{p\}$

    end if

  end for

  if $|\mathcal{I}| > d$ **then**

    $(c_\mathcal{I}, \mathbf{n}_\mathcal{I}, err_\mathcal{I}) \leftarrow$ FITPLANESVD($\mathcal{I}$)

    if $err_\mathcal{I} < err$ **then**

      $c^{best} \leftarrow c_\mathcal{I}$

      $\mathbf{n^{best}} \leftarrow \mathbf{n}_\mathcal{I}$

      $besterr \leftarrow err_\mathcal{I}$

    end if

  end if$it \leftarrow it + 1$

end while

return (c^{best}, $\mathbf{n^{best}}$)

4.2.2 Hand direction estimation

In this work the hand direction, denoted by $\mathbf{x_h}$, is estimated as the first component of Principal Component Analysis (PCA) applied to the 3D points in $\mathcal{H}$, which roughly corresponds to the vector going from the wrist to the fingertips.

Note that the direction computed in this way is not very precise and depends on the position of the fingers in the performed gesture. It gives, however, a general indication of the hand orientation. Moreover, the estimated $\mathbf{x_h}$ could, instead, be directed from the fingers to the forearm, thus leading to wrong assumptions in the next steps of the recognition pipeline.

Again, as for the correction of palm direction, tracking information from previous frames (if available) can be exploited to reliably assert the correctness of the estimated hand orientation. Whenever this kind of information is not available, further assumptions on the gesture set may be used as criteria. For example, if fingers are never expected to point downwards, an estimated axis $\mathbf{x_h}$ pointing to the ground surely means the method estimated the right orientation but the wrong direction.

In order to build a 3D coordinate system centered on the palm centroid C_P previously defined, the axis $\mathbf{x_h}$ is then projected on the estimated palm plane π (Eq. 4.10). This operation compensates for the possible orientation error introduced by the partially folded fingers, as their 3D points are taken into account for the $\mathbf{x_h}$ computation.

$$\mathbf{x_p} = \mathbf{x_h} - [\mathbf{x_h}^T \mathbf{z_p}]\mathbf{z_p} \tag{4.10}$$

with $\mathbf{x_p}$ the projection of $\mathbf{x_h}$ on π. Note that $\mathbf{x_p}$ and $\mathbf{z_p}$ are orthogonal by definition. The missing axis $\mathbf{y_p}$ is obtained by the cross-product of $\mathbf{z_p}$ and $\mathbf{x_p}$ thus forming a right-handed reference system $(\mathbf{x_p}, \mathbf{y_p}, \mathbf{z_p})$.

Finally, note also that C_P does not necessary lie on π, e.g. it could lie on a finger folded over the palm. In order to place C_P closer to the actual hand center, the point is projected on π by Eq. 4.10. The complete hand coordinate system is depicted in Fig. 4.8.

4.3 Hand segmentation

The proposed framework has, so far, gathered all the necessary information required for the hand partitioning in its relevant parts: $\mathcal{P}$, recalling it was defined

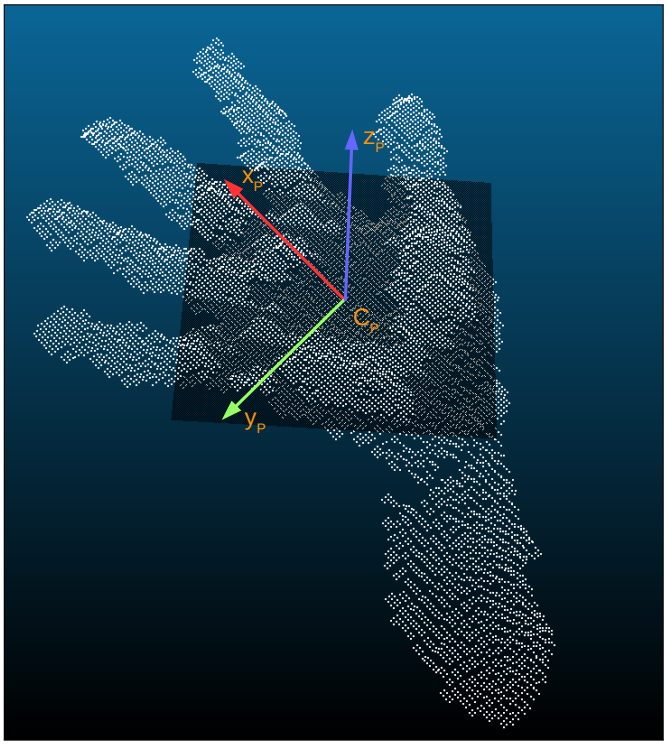


Figure 4.8: Estimated hand local reference system $(\mathbf{x_p}, \mathbf{y_p}, \mathbf{z_p})$

as the subset of the hand samples in $\mathcal{H}$ belonging to the palm only, was computed by Eq. 4.5 or Eq. 4.6. It follows that $\mathcal{H} \setminus \mathcal{P}$ is the set of hand samples belonging either to the fingers or to the first part of the forearm. Knowing the palm coordinate system of Fig. 4.8 and the palm parameters is now enough to discriminate the finger samples from the forearm ones.

Assume, for clarity sake, the palm has been detected by the circle fitting algorithm of Section 4.1.1, with r_p the estimated palm radius and C_P the estimated palm center projected on the palm plane π. Let $\mathbf{X_i^P} = [x_i^p \quad y_i^p \quad z_i^p]^T$ denote the coordinates of a generic hand sample $X_i \in \mathcal{H}$ expressed on the palm 3D coordinate system, obtainable by the simple transform in Eq. 4.11.

$$\mathbf{X_i^P} = R\mathbf{X_i} + \mathbf{t} \tag{4.11}$$

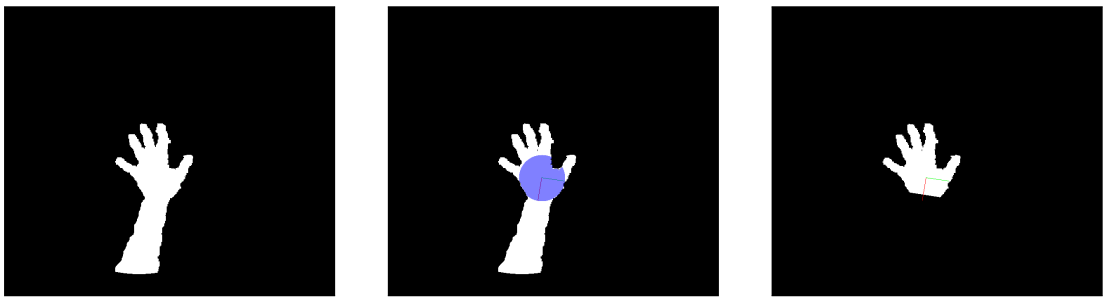
where R denotes the rotation matrix between the palm and the world (depth camera) coordinate systems and $\mathbf{t}$ their translation. Note how $\mathbf{t}$ simply corresponds to the palm centroid C_P, and R can be directly built from the palm coordinate system axis ($R = [\mathbf{x_p} \quad \mathbf{y_p} \quad \mathbf{z_p}]$).

Let now R_P denote the palm radius in the 3D space, obtained by the back-

projection of r_p (Eq. 3.2).It is possible to assume that X_i^P belongs to the wrist/forearm point cloud $\mathcal{W}$ whenever $x_i^p < -R_P$ (recall that the hand x axis points from the palm center to the fingertips as shown in Fig. 4.8).

Note how the estimated radius R_P is not accurate and may lead either to some palm points removal or to retain some wrist points. A more accurate approach consists in performing the wrist removal in the hand depth mask B_H analogously to the first method. Firstly PCA is performed on B_H to estimate the hand direction in the image plane and, jointly to the palm centroid c_p, to define an hand 2D local coordinate system. Then all the hand point coordinates in B_H are expressed in the 2D hand coordinate system and, this time, all the transformed points with u_P value higher than r_p are considered to belong to the forearm. The wrist removal accuracy is higher since the detected palm in B_H is rather reliable. In case of ellipse fitting, r_p has to be replaced by the u_p value of the intersection point of the ellipse with the negative semi-u_p-axis.

The novel depth mask B_A obtained from the arm removal in B_H is at the base of several feature extraction algorithms of Chapter 5.



(a) Detected hand binary mask prior arm removal

(b) Detected palm and hand coordinate system

(c) Detected hand binary mask after arm removal

Figure 4.9: Example of arm removal on a binary mask

After computing $\mathcal{W}$, the finger samples set $\mathcal{F}$ may be obtained again with a simple binary set operation, as reported in Eq. 4.12.

$$\mathcal{F} = \mathcal{H} \setminus (\mathcal{P} \cup \mathcal{W}) \tag{4.12}$$

Finally, it is useful to define, for feature extraction purposes, another sample set $\mathcal{H}_P$ made by the union of the palm samples with the finger ones (Eq. 4.13).

$$\mathcal{H}_P = \mathcal{H} \setminus \mathcal{W} = \mathcal{P} \cup \mathcal{F} \tag{4.13}$$

Chapter 5

Feature extraction

Current chapter describes with an higher level of detail what features the proposed framework allows the extract from the segmented data of Chapter 4 and how this task is performed.

In this context, the features describe geometrical, textural or other kind of hand properties that are *quantifiable* and *robust*, that is, they must assume similar values in equal conditions. For example, if a feature describes the length of a finger in a particular gesture, this value must not change sensibly whenever the user performs the same gesture in different instants.

It is important to recall that, while the previous steps in the gesture recognition pipeline are rather common among the approaches in literature of Section 1.2, the selected feature sets and their extraction algorithms are generally peculiar of the different methods. Different descriptors extracted from the same data may lead to noticeable changes in the recognition accuracy of a given machine-learning approach, as will be shown by the results of Chapter 7.

For this reason, most of the research in this field has been devoted to the discovery of new hand features to extract and the improvement of the existing ones. In particular, this thesis shows how depth information allows the extraction of robust features describing the hand 3D geometry and how certain feature extraction algorithms in literature designed for the recognition of objects different from hands can be adapted for the gesture recognition purposes.

The proposed framework implements several extraction algorithms for feature extraction belonging to two families: geometrical features, describing 2D or 3D properties of the fingers or the palm, and color features, describing textural properties of the segmented hand from the background.

Geometrical features are either extracted from the acquired depth map or the 3D points computed by its back-projection (Eq. 3.2), and include:

- **Hand contour distances from the palm center:** describe the Euclidean distances of the fingertips from the estimated palm center C_P. They may be extracted from the 3D points of the finger set $\mathcal{F}$ or associated to the hand boundary in the binary depth mask B_A computed from the set $\mathcal{H}_\mathcal{P}$.

- **Hand contour distances from the palm plane:** describe the Euclidean distances of the fingertips from the estimated palm plane π. They are extracted from the 3D points of the finger set $\mathcal{F}$.

- **Hand contour similarities:** compare quantitatively the hand contour of the performed gesture with the one of each gesture template in the selected gesture dictionary in order to detect the most similar.

- **Hand contour curvatures:** aim at cataloging each gesture according to the number of the convexities and concavities detected on the hand contour in the binary depth mask built from the set $\mathcal{H}_\mathcal{P}$.

- **Palm morphology features:** describe the shape of the palm region and help to state whether each finger is raised or folded on the palm according to the flatness of the palm surface.

- **Convex hull features:** quantify several differences between the hand shape in the depth mask B_A and the related *convex hull*. They include the ratios between the area or the perimeter of the hand shape and the one of its convex hull, the number of the convex hull vertexes and the number and the sizes of the regions between the fingers.

- **Fingertip orientations:** measure the angle formed by the segments joining each detected fingertip $F_j, j = 1, \ldots, 5$ projected on the palm plane π with the palm center C_P and the hand direction $\mathbf{x_p}$.

- **Fingertip positions:** characterize a gesture according to the positions of each detected fingertip in the 3D space, expressed in the local hand coordinate system $(\mathbf{x_p}, \mathbf{y_p}, \mathbf{z_p})$ pinned on C_P.

It is worth noting that the previously described features require the presence of a range camera in the acquisition setup, since they are extracted from a depth map. Another set of geometrical features that can be extracted comes from the data provided by the Leap Motion software and, differently from the previous case, it does not require the processing of a depth map. The features include:

- **Fingertip distances from the palm center or the palm plane:** describe the same properties of the related features extracted from the hand depth map, but exploiting the 3D fingertip positions returned by the Leap Motion APIs.
- **Fingertip orientations and positions:** describe the same properties of the related features extracted from the hand depth map, but exploiting the 3D fingertip positions returned by the Leap Motion APIs.
- **Inter fingertip distances:** measured between pairs of detected fingertips, help to discriminate gestures showing the same number of raised fingers and similar finger lengths but a different finger arrangement.
- **Hand radius:** exploits the hand radius returned by the Leap Motion APIs to discriminate gestures only differing for the closeness status of the fingers.
- **Number of detected fingers:** prevent the misclassification of the performed gesture with another one with a different number of raised fingers.

In case of an acquisition setup made by a Leap Motion and a range camera, the proposed framework also offers a more performant set of algorithms for the extraction of most geometrical feature sets exploiting both depth information and the Leap Motion data.

Finally, the considered textural features are:

- **Histogram of oriented gradients (HOG):** based on the idea that the local shape around an hand point can be described rather well by the distribution of local intensity gradients.
- **Local phase quantization:** characterizes the hand shape with the distribution of the local phase of the Fourier transform around each hand point.
- **Local ternary patterns:** encode the differences between each hand pixel and the surrounding ones within a limited size window.

The proposed color features can be extracted either from the acquired color image or from the depth map $D = \{d_{u,v}\}$ represented as a grayscale image $I_G = \{i^G_{u,v}\}$ with the simple transform of Eq. 5.1:

$$i^G_{u,v} = I^{max}_G \frac{d_{u,v} - D^{min}}{D^{max} - D^{min}} \tag{5.1}$$

where I_G^{max} denotes the maximum gray value (e.g., $I_G^{max} = 1$ for grayscale image pixels expressed with floating point values or $I_G^{max} = 255$ for grayscale image pixels expressed with integer values) and D^{min}, D^{max} the minimum and maximum measurable depth values by the employed range camera.

5.1 Depth data features

This section describes with an high level of detail all the geometrical features listed in the beginning of the current chapter.

5.1.1 Hand contour distances from the palm center

Hand contour distances from the palm center belong to the family of features describing the performed gesture on the basis of the hand shape variations. The extraction of this feature set starts from the construction of a plot representing the maximum distances of the edge samples in $\mathcal{F}$ from the hand centroid C_P, under the rationale that the shape of the plot is characteristic for each different gesture [54, 55, 56].

The method extends the main idea of [11] by exploiting Euclidean distances in the 3D space instead of the pixel distances in the image plane, thus preventing the loss of information due to the projective geometry of pinhole model (Eq. 2.1).

Let R_P denote, again, the computed 3D radius in Section 4.3 for the arm removal. For each 3D point $X_i \in \mathcal{F}$, the algorithm computes its normalized distance from the centroid $d(X_i)$ and the angle $\theta(X_i)$ between the vector $\mathbf{X}_i^{\pi} - \mathbf{C_P}$ and axis $\mathbf{x_p}$ on the palm plane π, according to Eq. 5.2.

$$\begin{aligned} d(X_i) &= \begin{cases} \frac{\|\mathbf{X_i} - \mathbf{C_P}\| - R_P}{d^{max}} & \text{if } \|\mathbf{X_i} - \mathbf{C_P}\| - R_P \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ \theta(X_i) &= \angle(\mathbf{X}_i^{\pi} - \mathbf{C_P}) = 2 \arctan\left(\frac{\sqrt{x_p^2 + y_p^2} - y_p}{x_p}\right) \end{aligned} \tag{5.2}$$

where $\mathbf{X_i^{\pi}}$ denotes the projection of X_i on the plane π, d^{max} is the maximum distance from a generic finger sample X_i and the hand centroid C_P (which usually corresponds to the length of the mean finger), and x_p, y_p, z_p are the coordinates of $\mathbf{X_i^{\pi}}$ expressed in the hand coordinate system. Note how the arc tangent in Eq. 5.2 sometimes used in high precision computation and is defined for the whole trigonometric circle.

The range $[0, 360)$ of the possible values for $\theta(X_i)$ is then sampled with a uniform quantization step Δ_q into a discrete set of values $\{\theta_1^q, \ldots, \theta_M^q\}$ with $M = \lfloor 360/\Delta_q \rfloor$ (e.g. in the results of Chapter 7 Δ_q has been set to 2°). Each θ_j^q thus represents the angular sector $\mathcal{I}_j^q = [\theta_j^q - \frac{\Delta_q}{2}, \theta_j^q + \frac{\Delta_q}{2}]$. The quantized value of $\theta(X_i)$, denoted by $\theta^q(X_i)$, is obtained by Eq. 5.3.

$$\theta^q(X_i) = \left\lfloor \frac{\theta(X_i)}{\Delta_q} \right\rfloor + \frac{\Delta_q}{2} \tag{5.3}$$

Let now $\mathcal{X}_j^q$ denote the set of points $X_i \in \mathcal{F}$ whose quantized angle $\theta^q(X_i)$ value is θ_j^q. The feature extraction algorithm builds a plot $L(\theta_q)$ reporting for each angular value θ_j^q the maximum distance $d(X_i)$ of the points within $\mathcal{X}_j^q$, namely the points "falling" in the same angular sector $\mathcal{I}_j^q$ (Eq. 5.4).

$$L(\theta_j^q) = \max_{X_i \in \mathcal{X}_j^q} d(X_i) \tag{5.4}$$

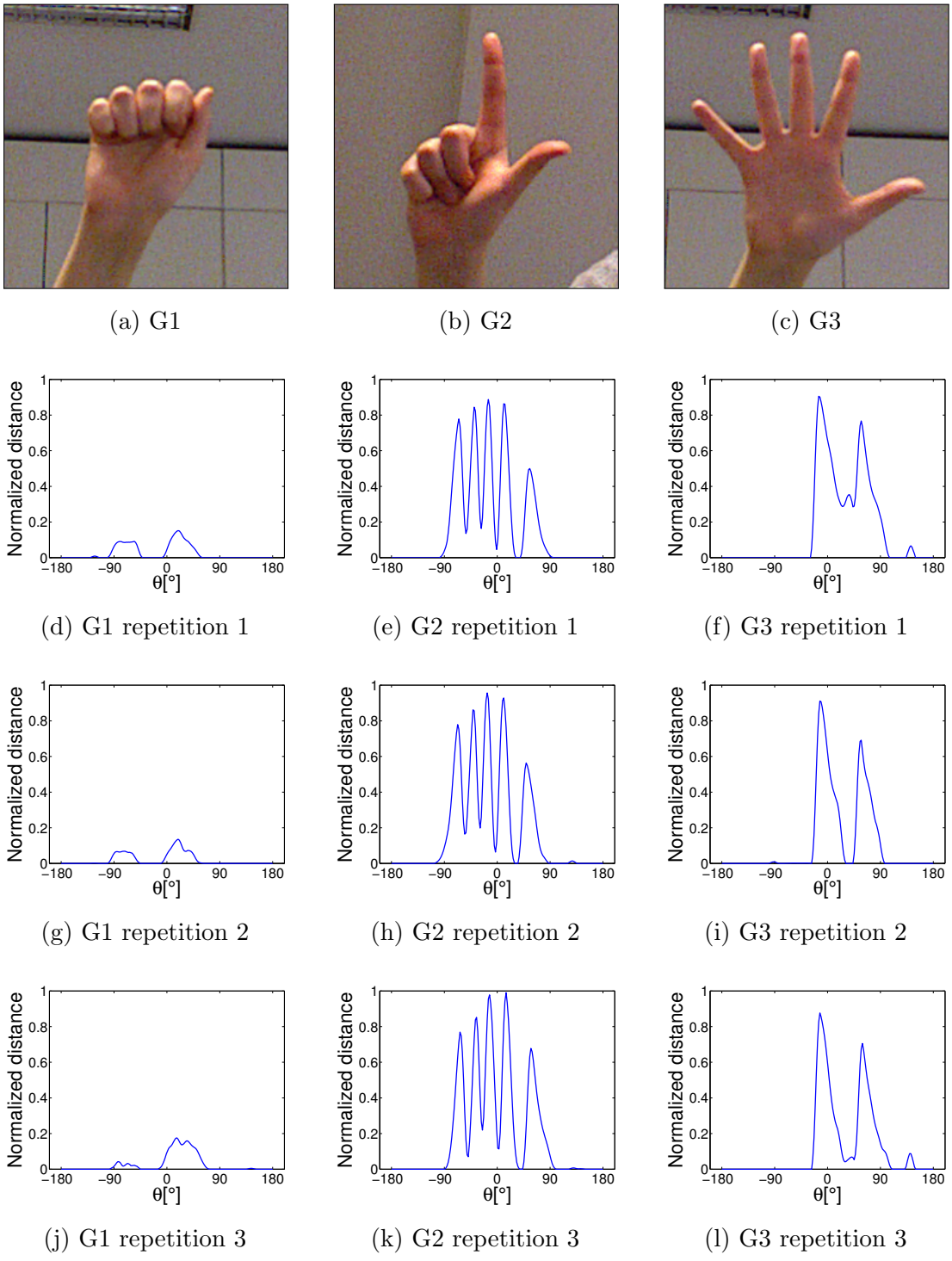
$L(\theta_q)$ is then smoothed by the convolution with a gaussian kernel of short support in order to minimize the detrimental effects of the finger samples noise and favor the following steps in the feature extraction pipeline. An example of generated plots by Eq. 5.4 for a few gestures is shown in Fig. 5.1.

Although the shape of $L(\theta_q)$ characterizes the performed gesture and the scaling by d^{max} in Eq. 5.2 potentially allows the comparison of plots referred to gestures of the same user or different users, $L(\theta_q)$ cannot be directly used as a feature vector describing the fingers outline in the 3D space. The reason is the strong dependence of $L(\theta_q)$ from the hand orientation that, besides being ratherly inaccurate, does not allow to map the angular sectors to the same fingers in different gestures or even in different repetitions of the same gesture.

For this reason, the proposed feature extraction algorithm compensates the systematic error in the hand main direction estimation by aligning $L(\theta_q)$ with a reference plot $L_g^r(\theta_q)$ generated in the calibration phase for each gesture g of the dictionary.

The alignment of $L(\theta_q)$ with a gesture template $L_g^r(\theta_q)$ consists in looking for the translational shift (that is actually a modulus shift) ϕ_g^r of $L(\theta_q)$ maximizing the value of a similarity metric $\rho(\cdot)$ between $L(\theta_q + \phi_g^r)$ and $L_g^r(\theta_q)$ (Eq. 5.5).

$$\phi_g^r = \underset{\phi}{\operatorname{argmax}}\, \rho\left(L(\theta_q + \phi), L_g^r(\theta_q)\right) \tag{5.5}$$



(a) G1 (b) G2 (c) G3

(d) G1 repetition 1 (e) G2 repetition 1 (f) G3 repetition 1

(g) G1 repetition 2 (h) G2 repetition 2 (i) G3 repetition 2

(j) G1 repetition 3 (k) G2 repetition 3 (l) G3 repetition 3

Figure 5.1: Comparison of maximum distances from the palm center

A common robust metric employed in pattern recognition for this purpose is the *cross-correlation* between $L(\theta_q)$ and $L_g^r(\theta_q)$. The proposed framework exploits, indeed, a variation named *zero-mean normalized cross-correlation* (ZNCC) (Eq. 5.6) due to its capability of correctly matching $L(\theta_q)$ and $L_g^r(\theta_q)$ when their maximum amplitudes are different. The classic definition of cross-correlation and other variations like the *sum of squared distances* (SSD) reported, in fact, very low performance because of the sensibility to the plot amplitude in the first case and the amplification of noise in the second.

$$\rho_z(L(\theta_q), L_g^r(\theta_q)) = \frac{\sum_{\theta_q=0}^{\lfloor 360/\Delta_q \rfloor} (L_g^r(\theta_q) - \overline{L_g^r}(\theta_q))(L(\theta_q) - \overline{L}(\theta_q))}{\sqrt{\sum_{\theta_q=0}^{\lfloor 360/\Delta_q \rfloor} (L_g^r(\theta_q) - \overline{L_g^r}(\theta_q))^2 \sum_{\theta_q=0}^{\lfloor 360/\Delta_q \rfloor} (L(\theta_q) - \overline{L_g^r}(\theta_q))^2}} \tag{5.6}$$

where $\overline{L}(\theta_q)$ and $\overline{L_g^r}(\theta_q)$ denote the arithmetic means of the two plots.

The implemented alignment procedure consists, then, in looking for the translational shift maximizing the ZNCC between the translated version of $L(\theta_q)$ and the reference plot $L_g^r(\theta_q)$.

Recall from Section 4.2.1 that, depending on the application of gesture recognition, the palm orientation is always assumed to point towards the acquisition setup or, conversely, the dictionary either accounts for gestures with palm or dorsum facing it. Furthermore, a more flexible application may also allow the user to perform a selected gesture either with the palm or the dorsum facing the acquisition setup. While in the first case the generated distance plots implicitly induce a finger ordering, in the remaining cases the finger ordering in the plots depends on the $\mathbf{z_p}$ axis direction. It follows that in certain situations a plot aligned with its template presents a lower similarity value respect to the same plot aligned with the template of another gesture only because the aligned plot is "flipped" respect to the template abscissa.

In order to solve this problem, Eq. 5.7 extends Eq. 5.5 considering the possibility of flipping $L(\theta_q)$ before the alignment with a given gesture template $L_g^r(\theta_q)$.

$$\begin{aligned} \phi_g^r &= \operatorname*{argmax}_{\phi} \rho_z \left(L(\theta_q + \phi), L_g^r(\theta_q) \right) \\ \phi_g^{r,rev} &= \operatorname*{argmax}_{\phi} \rho_z \left(L(-\theta_q + \phi), L_g^r(\theta_q) \right) \end{aligned} \tag{5.7}$$

It is worth noting that ϕ_g^r may be different not only among each gesture template, but also among repetitions of the same gesture in different instants, thus compensating the limited accuracy of the direction computed by the PCA. The alignment procedure solves one of the main issues related to the direct application of the approach of [11].

The distance plot $L(\theta_q)$ aligned to the gesture template $L_g^r(\theta_q)$, denoted by $L_g(\theta_q)$, is then computed with Eq. 5.8.

$$L_g(\theta_q) = \begin{cases} L(\theta_q + \phi_g^r) & \text{if } \rho_z\left(L(\theta_q + \phi_g^r), L_g^r(\theta_q)\right) \geq \rho_z\left(L(-\theta_q + \phi_g^{r,rev}), L_g^r(\theta_q)\right) \\ L(-\theta_q + \phi_g^{r,rev}) & \text{otherwise} \end{cases} \tag{5.8}$$

Let now G be the number of different gestures in the considered dictionary. Let also $\mathcal{I}_{g,j}^r(\theta_q) = \theta_{g,j}^{min} < \theta_q < \theta_{g,j}^{max}$ for $j \in \{1,..,5\}$ denote the angular interval associated to the j-th raised finger in each gesture template $g \in \{1,..,G\}$. The 3D distances descriptor is made by a juxtaposition of the aligned distance plot peaks within the previously defined angular regions in each gesture template, namely, the feature value $f_{g,j}^l$ associated to finger j in gesture g corresponds to the maximum of the aligned scaled distance plot within the angular region $\mathcal{I}_{g,j}^r(\theta_q)$.

$$f_{g,j}^l = \max_{\mathcal{I}_{g,j}^r(\theta_q)} L_g(\theta_q) \tag{5.9}$$

An example of extracted distance 3D features from the alignment of the computed $L(\theta_q)$ with a given gesture template $L_g^r(\theta_q)$ is shown in Fig. 5.2.

Note how the descriptor can contain up to $G \times 5$ features, although their actual number is smaller since not all the fingers are raised in each gesture and the raised finger regions are the only ones of interest. The distance features are collected into feature vector $\mathbf{F}_1^l$.

A more recent version of this descriptor avoids the alignment of the distance plot $L(\theta_q)$ with each gesture template $L_g^r(\theta_q)$, by exploiting the outline of the hand contour on the range camera image plane and a proper ordering of the hand contour points [57].

Let B_A denote the binary mask built on the set $\mathcal{H}_P$ of Eq. 4.13, that is, the binary mask only selecting the hand points after the segmentation of Chapter 4 and the wrist removal. Let also $\partial\mathcal{H}_P$ denote the *frontier* of $\mathcal{H}_\mathcal{P}$, namely the subset

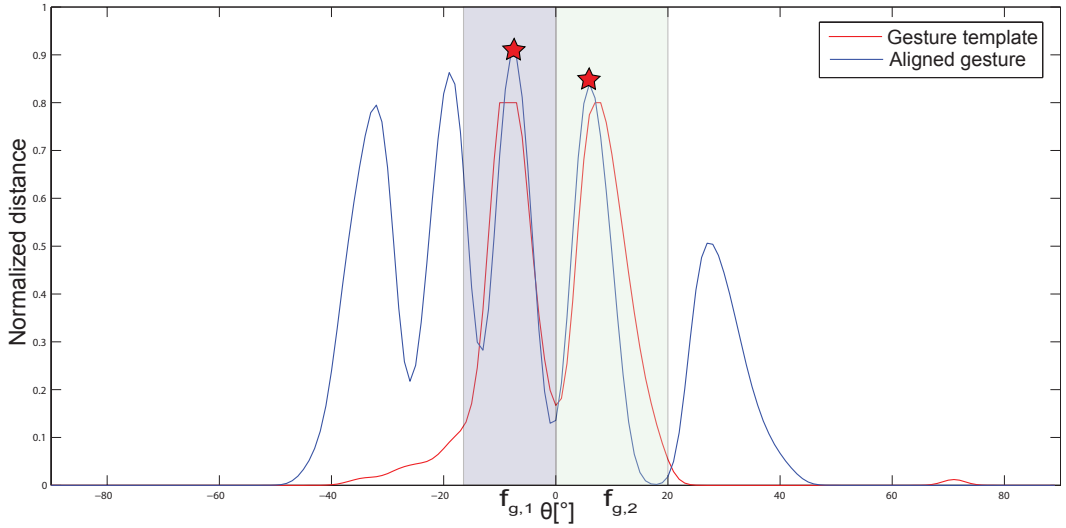


Figure 5.2: Example of extracted peaks from the alignment with a gesture template

of the samples of $\mathcal{H}_\mathcal{P}$ corresponding to the pixels of the B_A contour computed with a state-of-art edge detector (e.g., Canny [51]).

The extraction of the current descriptor starts by constructing a mono-dimensional representation of the normalized distances of the 3D points $P_i \in \partial\mathcal{H}_P, i = 1, \ldots, |\partial\mathcal{H}_P|$ from the estimated hand centroid C_P (Eq. 5.10).

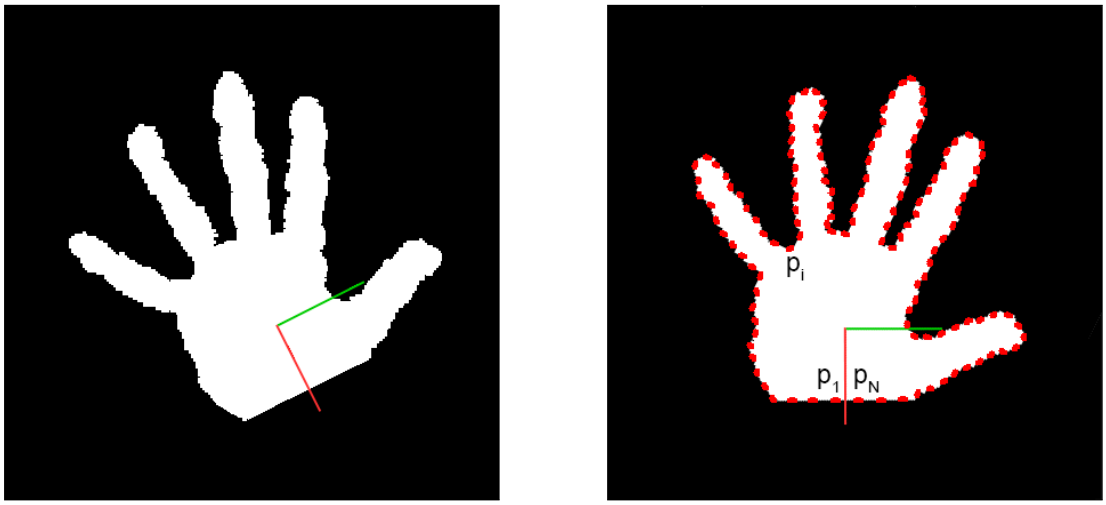
$$L(p_i) = \frac{\|\mathbf{P_i} - \mathbf{C_P}\|}{L^{max}} \tag{5.10}$$

where p_i denotes the projection of the hand contour point P_i on the image plane of the range camera and L^{max} the length of the longest finger (e.g., mean finger) in order to scale the plot values in the range $[0, 1]$.

It is worth noting that the plot generated from Eq. 5.10 cannot be directly used as a distance descriptor since the contour pixels p_i do not refer to the same hand contour points for different gesture repetitions. Furthermore, the different lengths of the hand contour in each frame would lead to the creation of varying size feature vectors, which are unsuitable for classification purposes.

In order to reestablish an ordering among the hand contour pixels, the algorithm first exploits the main hand direction in B_A and the centroid c_p to *rectify* B_A, namely to rotate the binary mask for aligning the main hand direction with the v axis of the image plane. Then, the method indexes the hand contour pixels starting from the one lying on intersection of the hand contour with the rectified

hand main axis, roughly corresponding to the the center of the wrist. An example of rectification is reported in Fig. 5.3.



(a) Original hand mask (b) Rectified hand mask

Figure 5.3: Example of depth mask rectification and hand contour pixels indexing

Let $p_j, j = 1, \ldots, N$ denote the renumbered hand contour pixels and P_j the associated 3D points by back-projection (Eq. 3.2). Analogously to Eq. 5.2, the descriptor now computes an hand contour plot with Eq. 5.10 and smooths it with a Gaussian filtering. Finally, the plot is sampled uniformly in order to always retain the same number $M = N/K$ of distance values, where K divisor of N denotes the quantization step.

Fig. 5.4 shows an example of generated plot by the previously described algorithm. Note how the plot, again, characterizes the performed gesture.

The returned feature descriptor, this time, is directly the plot resulting from the previous processing, and not the juxtaposition of a few extracted distance peaks from the plot alignment with each gesture template. The lack of need for a template alignment in this case is due to the rectification and edge pixels indexing described in the previous rows, but the new feature vector of length M, denoted by $\mathbf{F_2^l}$, is usually several magnitude orders longer than the feature vectors generated by previous version of the descriptor when the cardinality of the gesture dictionary is rather low.

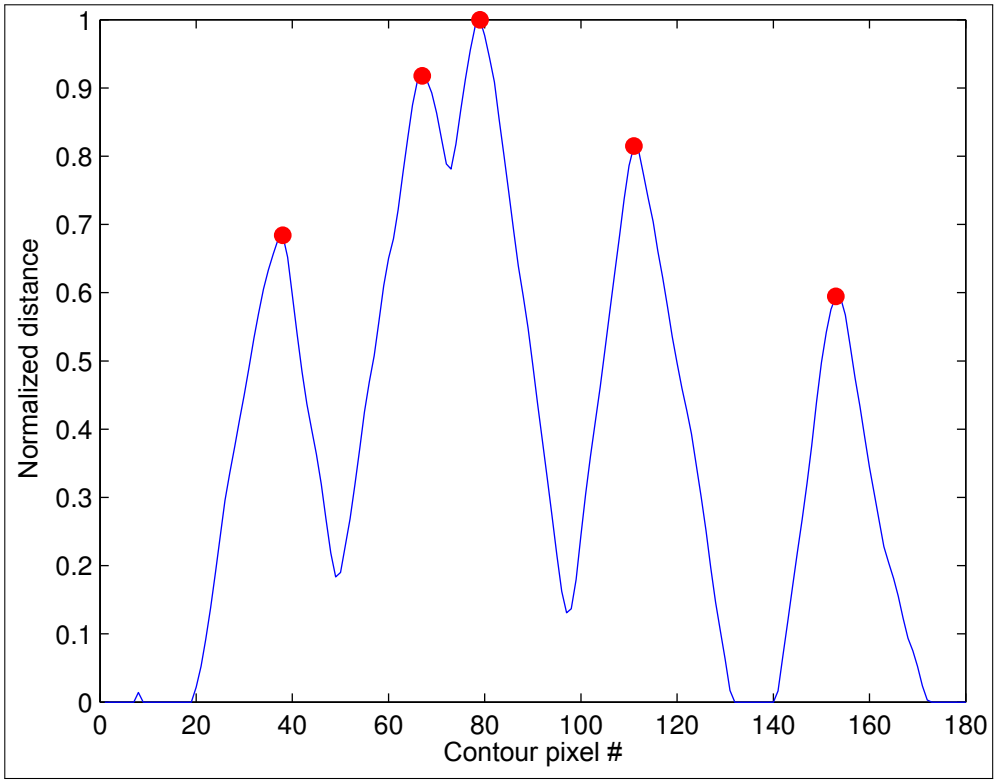


Figure 5.4: Example of generated distance plot from the hand contour points

5.1.2 Hand contour distances from the palm plane

Analogously to the descriptor of Section 5.1.1, gestures may also be characterized by the distances of the finger samples from the palm plane π [54, 56]. The rationale is that the considered gesture dictionary may contain gestures sharing similar distances of the fingertips from the palm center C_P but different fingertip positions in the 3D space, a property not captured by the descriptor of Section 5.1.1.

Let $e(X_i)$ denote the signed distance of the 3D point $X_i \in \mathcal{F}$ from the palm plane π, computed by Eq. 5.11.

$$e(X_i) = \text{sgn}\left((\mathbf{X_i} - \mathbf{X_i^\pi}) \cdot \mathbf{z_p}\right) \|\mathbf{X_i} - \mathbf{X_i^\pi}\| \tag{5.11}$$

where X_i^π denotes the projection of X_i on the palm plane π. The sign of $e(X_i)$ accounts for the fact that X_i can belong to any of the two semi-spaces defined by π, that is, X_i can either be on the front or behind π.

The current feature extraction consists in a simple adaption of the algorithm described in Section 5.1.1: first, a plot $E(\theta_q)$ representing the signed distance of each sample X_i in $\mathcal{F}$ from the palm plane π is built, analogously to the distance plot of Eq. 5.4. Then, $E(\theta_q)$ is aligned to each gesture template and a set of fingertip distances from the palm plane is extracted from the selected regions within the aligned plots.

$$E(\theta_j^q) = \frac{1}{L^{max}} \begin{cases} \max\limits_{X_i \in \mathcal{X}_j^q} e(X_i) & \text{if } \left| \max\limits_{X_i \in \mathcal{X}_j^q} e(X_i) \right| > \left| \min\limits_{X_i \in \mathcal{X}_j^q} e(X_i) \right| \\ \min\limits_{X_i \in \mathcal{X}_j^q} e(X_i) & \text{otherwise} \end{cases} \tag{5.12}$$

where θ_j^q denotes the sampled angle with the same quantization step used for the distance descriptor of Section 5.1.1 and L^{max} the longest finger length as scale factor. Fig. 5.5 shows an example of the elevation plot for a few gestures.

It is worth noting that, while the distance plots of Fig. 5.1 are rather robust respect to the same gesture repetitions, the plots generated by Eq. 5.12 are strongly affected by the reliability of the plane fitting and may thus sensibly differ also in case of repetitions of the same gesture.

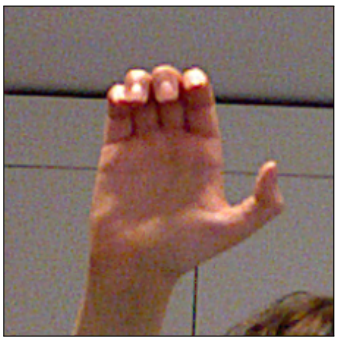
For this reason, the current descriptor avoids the alignment of $E(\theta_q)$ with each gesture template $E_g^r(\theta_q)$ and relies on the angular shifts previously computed in Section 5.1.1 for the alignment of $L(\theta_q)$ with the respective gesture templates $L_g^r(\theta_q)$.

Let $E_g(\theta_q)$ denote the plot $E(\theta_q)$ aligned with the g-th gesture template by a circular shift of ϕ_g^r. The distance features are then computed according to Eq. 5.13.

$$f_{g,j}^e = \begin{cases} \max\limits_{\mathcal{I}_{g,j}^r(\theta_q)} E_g(\theta_q) & \text{if } \left| \max\limits_{\mathcal{I}_{g,j}^r(\theta_q)} E_g(\theta_q) \right| > \left| \min\limits_{\mathcal{I}_{g,j}^r(\theta_q)} E_g(\theta_q) \right| \\ \min\limits_{\mathcal{I}_{g,j}^r(\theta_q)} E_g(\theta_q) & \text{otherwise} \end{cases} \tag{5.13}$$

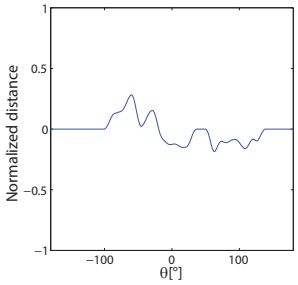
Finally, since the computation of $f_{g,j}^e$ is analogous to the one of $f_{g,j}^l$ (Eq. 5.9), the feature vector $\mathbf{F}^e$ made by the juxtaposition of the peaks $f_{g,j}^e$ extracted from each alignment of $E_g(\theta_q)$ with all the gesture templates has the same structure and number of elements of the vector $\mathbf{F}_1^l$ of Section 5.1.1.

(a) G1

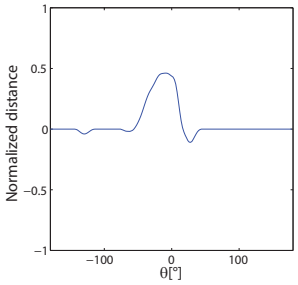


(b) G2

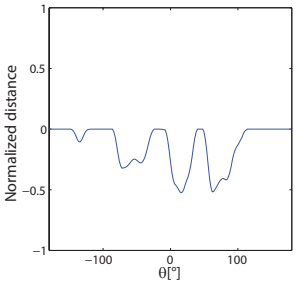
(c) G3



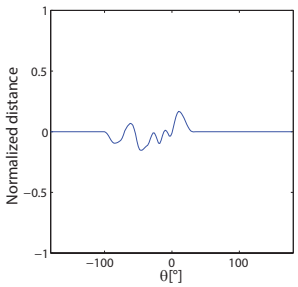
(d) G1 repetition 1



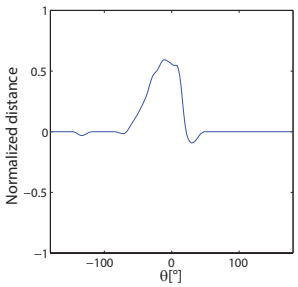
(e) G2 repetition 1



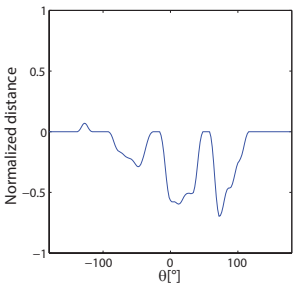
(f) G3 repetition 1



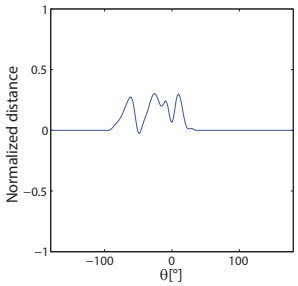
(g) G1 repetition 2



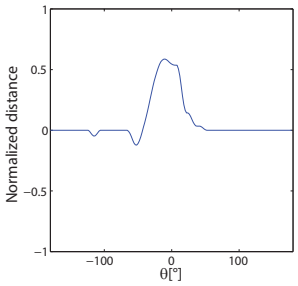
(h) G2 repetition 2



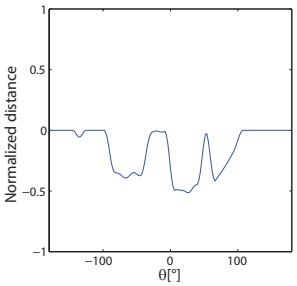
(i) G3 repetition 2



(j) G1 repetition 3



(k) G2 repetition 3



(l) G3 repetition 3

Figure 5.5: Comparison of maximum distances from the palm plane

5.1.3 Hand contour similarities

This feature set, directly extracted from the alignment data generated in Section 5.1.1, aims at discriminating the performed gesture according to the similarities of the hand outline respect to each gesture template [57, 58, 54].

Let $L(\theta_q)$ denote again the hand contour distances from the palm center plot of the performed gesture, $L_g^r(\theta_q)$ the reference distance plot for gesture g and $\rho_z^{g,max}$ the maximum correlation (ZNCC) value obtained from the alignment (Eq. 5.7).

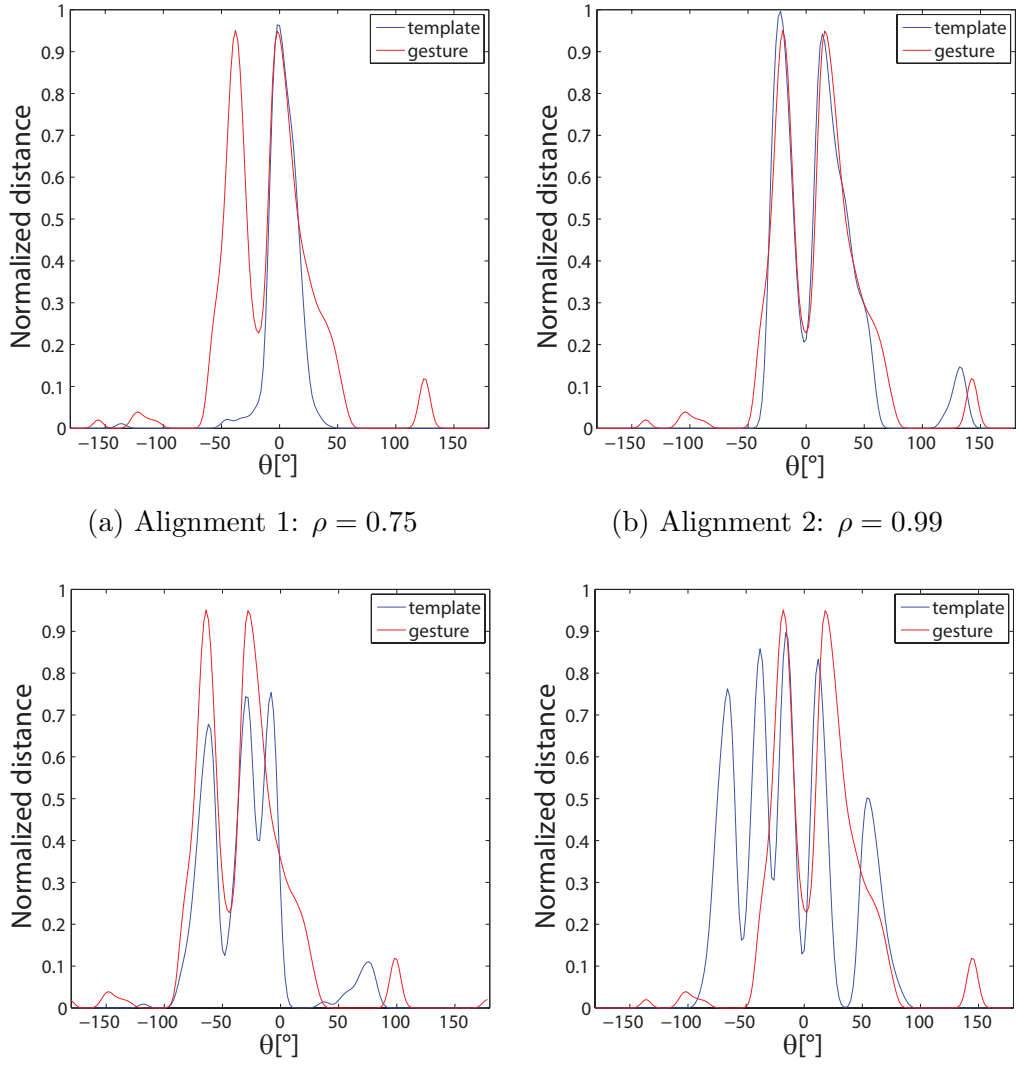
The feature vector for this descriptor, denoted by $\mathbf{F^z}$, is made by the juxtaposition of the maximum correlation $\rho_z^{g,max}$ for the alignment of $L(\theta_q)$ with every gesture template $L_g^r(\theta_q)$, analogously to the feature vectors of Sections 5.1.1 and 5.1.2. The rationale behind $\mathbf{F^z}$ is that, ideally, the alignment of $L(\theta_q)$ with the correct gesture template $L_g^r(\theta_q)$ returns the maximum correlation value among the ones returned by the alignment with the other templates. Fig. 5.6 shows an example of how the correlation value varies for the alignment of $L(\theta_q)$ with different gesture templates $L_g^r(\theta_q)$.

An important aspect of this feature extraction method is the similarity metric employed in Section 5.1.1, which both strongly affects the reliability of the gesture plot alignments and the maximum correlation value. While the chosen of *zero-mean normalized cross-correlation* (ZNCC) between the distance plots leads to better alignments, since this measure is less affected by the varying sizes of different hands, at the same time it may also be a weak point. ZNCC, in facts, sometimes is not able to discriminate two plots with a similar outline but different amplitudes: for example, a plot representing a gesture with a raised index finger only may have an high correlation value with another plot representing a gesture with a raised pinky finger only.

This ambiguity can be often removed by using alternative similarity measurement functions in place of or together with the ZNCC. A good alternative is the *sum of squared differences* (SSD) between the two aligned plots (Eq. 5.14).

$$\rho_s(L(\theta_q), L_g^r(\theta_q), \phi) = \frac{\sum_{\theta_q^1}^{\theta_q^M} \left[L_g^r(\theta_q) - L\left((\theta_q + \phi) \mod \left\lfloor \frac{2\pi}{\Delta_q} \right\rfloor \right) \right]^2}{S} \tag{5.14}$$

where $S = \lfloor 360/\Delta_q \rfloor$ is a scaling factor used to shift the correlation values in the range $[0, 1]$.



(a) Alignment 1: $\rho = 0.75$

(b) Alignment 2: $\rho = 0.99$

(c) Alignment 3: $\rho = 0.8$

(d) Alignment 4: $\rho = 0.7$

Figure 5.6: Comparison of the maximum correlation value for a few alignments

The SSD in this framework can be used in two different ways:

1. as a replacement of the ZNCC in Eq. 5.7.
2. As an integration of the ZNCC in order to overcome its limits.

While in the first case the SSD is best suited for the construction of a stand-alone feature vector $\mathbf{F}^s$, with usually worse performance than the ZNCC, the second option may be used, instead, to penalize alignments with templates with a different number of raised fingers or with templates with the same outline but different finger lengths. The extended descriptor $\mathbf{F}^\rho$ in this case is then made by juxtaposing $\mathbf{F^z}$ and $\mathbf{F^s}$.

5.1.4 Hand contour curvature

Hand contour curvature is another powerful descriptor of the hand shape which, alone, proved to be able to recognize the gestures contained in the datasets of Chapter 7 with a dramatically high accuracy [57, 58, 54, 55, 56]. This feature set, extracted on the hand depth mask B_A after the arm removal of Section 4.3, discriminates the different gestures on the basis of the detected concavities and convexities in the hand outline. The clenched hand, for example, is characterized by a overall convex shape, while the completely open hand (five co-planar fingers) is characterized by the concavities between consecutive fingers and the convexity around each fingertip.

Let $\mathcal{E}$ be a planar curve representing the hand contour in B_A. Let also $P \in \mathcal{E}$ be a generic point of $\mathcal{E}$. The *curvature* of $\mathcal{E}$ in P is a measure of how quickly the tangent line in P changes when moving to another point in its neighborhood. The curvature of a straight line is then, by definition, always 0 as the tangent over the line is constant, while the one of a circle with radius r is defined as $\kappa \triangleq 1/r$. $\kappa(P)$, that is, the curvature of $\mathcal{E}$ in P may then be geometrically defined as the the curvature of the *unique* circle that better approximates $\mathcal{E}$ around P, called the *osculating circle*. A pictorial example of this definition is shown in Fig. 5.7.

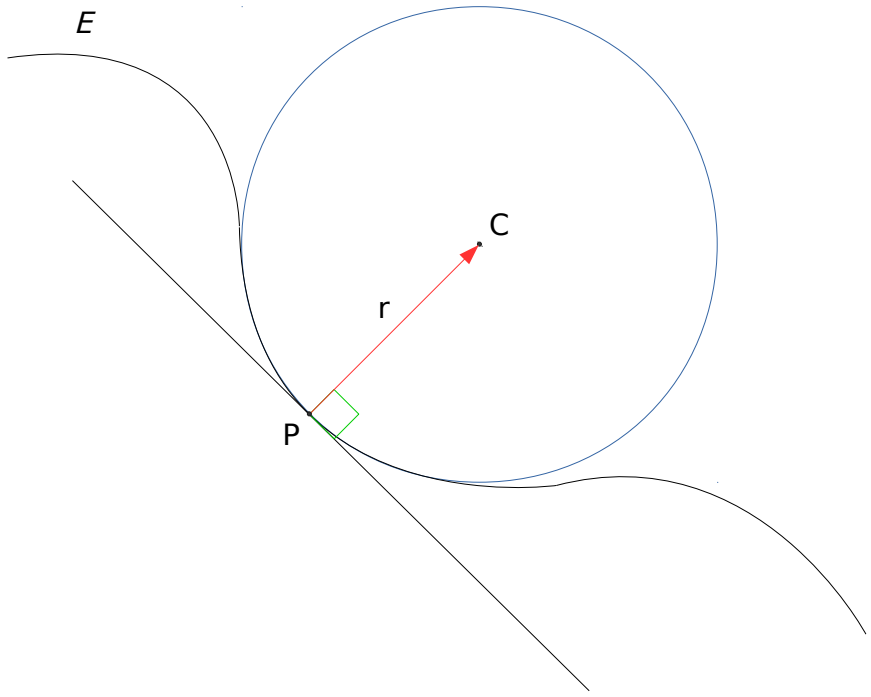


Figure 5.7: Osculating circle for $\mathcal{E}$ in P with center of curvature C and radius r

The previous classic definition of curvature may not, indeed, be exploited for the extraction of the current descriptor, as it requires a continuous parametric curve $\mathcal{E}$ and the usage of differential operators [59].

Hand contour in B_A is, instead, a discrete and non parametric curve described by the coordinates of the hand contour pixels. Moreover, since depth data coming from real-time range cameras are usually rather noisy, it is better to avoid differential operators for curvature description relying, instead, on integral invariants [60, 61].

The curvature descriptor presented in current section is inspired by the analogous 2D descriptor of [61] for leaf classification and improves it by removing its dependence on the object distance from the acquisition setup.

Consider a set of S circular masks $M_s(p_i), s = 1, \ldots, S$ of radius r_s centered on each edge pixel $p_i \in \partial B_A$, where ∂B_A denotes the hand contour extracted from B_A with edge detection techniques. For example, the tests of Chapter 7 used $S = 25$ and r_s varying from $0.5cm$ to $5cm$, where the radius r_s corresponds to the scale level. Note how circular masks were chosen due to their rotational invariance.

Let $C(p_i, s)$ denote the curvature in p_i at scale level s, expressed as the ratio of the number of samples of B_A within the mask $M_s(p_i)$ over $M_s(p_i)$ size, namely:

$$C(p_i, s) = \frac{\sum\limits_{M_s(p_i)} b^A_{u,v}}{|M_s(p_i)|} \tag{5.15}$$

where $b^A_{u,v} = 1$ if the depth sample with coordinates (u, v) is selected and $b^A_{u,v} = 0$ in the opposite case, and $|M_s(p_i)|$ denoting the cardinality of $M_s(p_i)$.

$C(p_i, s)$ is computed for each hand contour and mask scale s (Fig. 5.8).

Differently from [61] and other approaches, the radius r_s is defined in metrical units and is then converted by Eq. 5.16 to the corresponding pixel size on the basis of the depth of the hand centroid C_P . This expedient makes the descriptor invariant to the different sizes of the hand in B_A due to the projection on the range camera image plane of the hand moving in the 3D space.

$$r_s = \left\lfloor \frac{R_s}{2 z_{C_P} \frac{\tan\left(\frac{\pi f_H}{360}\right)}{D_W} + \frac{1}{2}} \right\rfloor \tag{5.16}$$

where R_s denotes the radius of mask M_s expressed in metrical units, z_{C_P} the depth of the estimated palm centroid, f_H and D_W the horizontal field of view

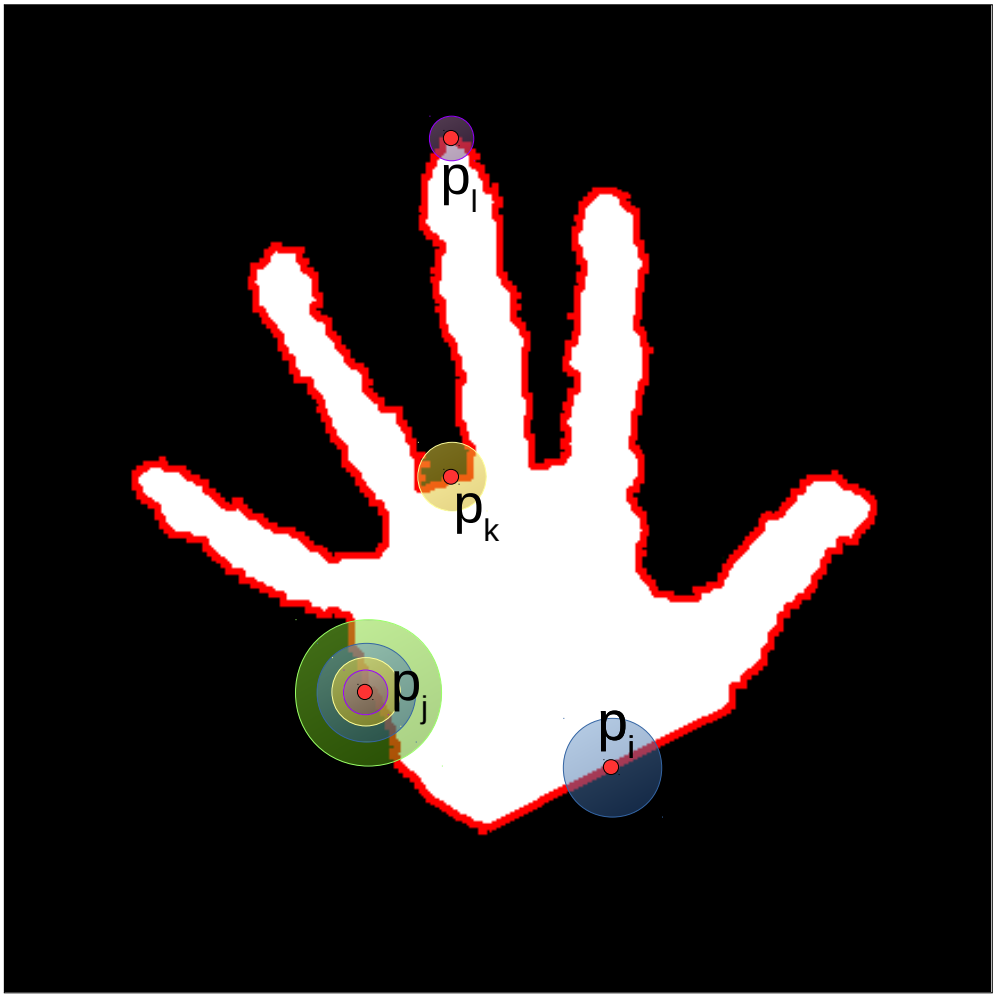


Figure 5.8: Example of curvature extraction with masks of varying size

and the depth map width of the employed range camera.

Although circular masks lead to extremely high recognition accuracies and are rotational invariant, the computational complexity of the curvature descriptor extraction increases dramatically with the number and size of the employed masks.

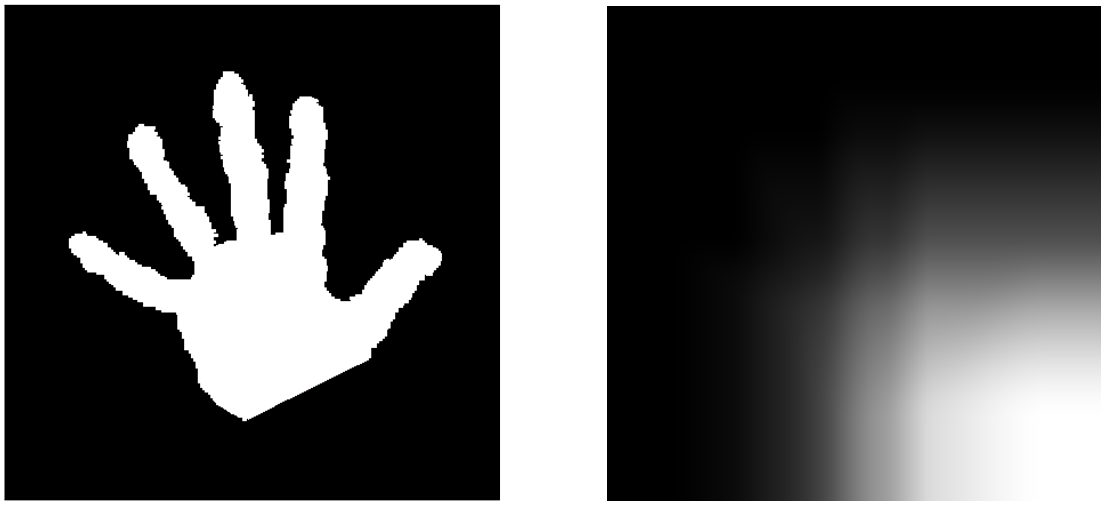
A possible way of reducing the computational load without affecting the descriptor accuracy consists in using an incremental approach exploiting *dynamic programming*. A simple *look-up-table* (LUT) indexed by the hand contour pixels, can, in facts, allow to reuse the curvature value computed for the previous mask size requiring only to evaluate the contribution of the pixels added by the current level mask. Note how the computational complexity of this improvement, although ways lower than the one of the naive approach, still depends on the employed mask sizes.

Another wiser variation of the basic approach, implemented in the proposed framework, requires only nearly one tenth of the processing time required by the naive method and introduces a neglectable loss in the descriptor accuracy. The main idea consists in relaxing the rotational invariance requirement by replacing the circular masks with square ones, and exploiting the *summed area table* algorithm, better known as *integral image* [8].

Let $I = \{i_{u,v}\}$ denote the integral image of the depth mask B_A, considered w.l.o.g. a binary image where the logical value "false" corresponds to 0 (black) and "true" to 1 or 255 (white). I is defined as:

$$i_{u,v} = \sum_{y=0}^{v} \sum_{x=0}^{u} b_{x,y}^{A} \tag{5.17}$$

Each pixel value in I is, then, the sum of all the pixels above and on the left of the pixel with coordinates (u, v) inclusive. Note how I can be computed in linear time with dynamic programming. An example of depth mask and the related integral image is shown in Fig. 5.9.



(a) Binary depth mask (b) Associated integral image

Figure 5.9: Example of binary depth mask and related integral image

The peculiarity of integral images is that the sum of the pixels over the rectangle spanned by $A = (u_0, v_0)$, $B = (u_0, v_1)$, $C = (u_1, v_0)$, $D = (u_1, v_1)$ with $u_0 \leq u_1$ and $v_0 \leq v_1$ in B_A, only requires a *constant time* if I is exploited (Eq. 5.18).

$$\sum_{u=u_0}^{u=u_1}\sum_{v=v_0}^{v=v_1} b_{u,v}^{A} = D + A - B - C \tag{5.18}$$

By setting the logical value "false" to 0 and "true" to 1 in B_A, it is straightforward to see that if A, B, C and D are the vertices of a square mask $M_s(p_i)$ centered on p_i in B_A, Eq. 5.18 exactly returns the number of hand samples within $M_s(p_i)$. Since the cardinality of $M_s(p_i)$ is known a priory, it is also known the density of hand points within $M_s(p_i)$, namely the curvature value $C(p_i, s)$ around p_i for the mask scale s defined previously.

The rotational invariance relaxation introduced by the usage of square masks only leads to a neglectable loss in recognition accuracy in front of a dramatic performance boost, thus making the integral version of the curvature descriptor suitable for real-time gesture recognition.

Now, both for circular and square masks, the values of $C(p_i, s)$ range from 0 (extremely convex shape) to 1 (extremely concave shape), with $C(p_i, s) = 0.5$ corresponding to the curvature of a straight edge. The $[0, 1]$ interval is quantized into N bins of equal size $b_1, \ldots, b_N$ (e.g., $N = 10$ for the tests of Chapter 7). Let now Eq. 5.19 define the set $\mathcal{C}_{j,s}$ of the hand contour pixels p_i having a curvature value $C(p_i, s)$ falling into the bin b_j for a curvature mask of radius r_s.

$$\mathcal{C}_{j,s} = \left\{ p_i \in \partial B_A | \frac{j-1}{N} \leq C(p_i, s) < \frac{j}{N} \right\} \tag{5.19}$$

For each radius value r_s and for each bin b_j the chosen curvature feature, denoted by $f_{j,s}^{c}$, is the cardinality of the set $\mathcal{C}_{j,s}$ normalized by the contour length $|\partial B_A|$ (Eq. 5.20).

$$f_{j,s}^{c} = \frac{|\mathcal{C}_{j,s}|}{|\partial B_A|} \tag{5.20}$$

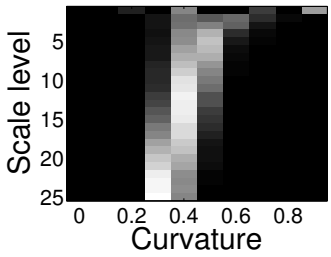
All the curvature features $f_{j,s}^{c}$ are collected in a feature vector $\mathbf{F}^c$ with $N \times S$ entries, ordered by increasing values of indexes j and s. Thanks to the normalization in Eq. 5.20, the curvature features $f_{j,s}^{c}$ only take values in the same range $[0, 1]$ shared by several other descriptors for comparison purposes.

By reshaping $\mathbf{F}^c$ into a matrix with S rows and N columns and by considering each $f_{j,s}^{c}$ as the intensity value of the pixel with coordinates (j, s) in a grayscale image, it is possible to graphically visualize the overall curvature descriptor $\mathbf{F}^c$ as exemplified in Fig. 5.10.

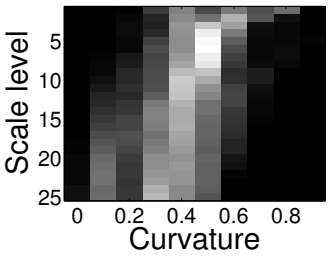
(a) G1

(b) G2

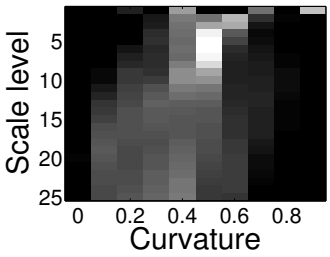
(c) G3



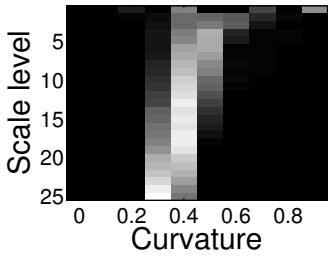
(d) G1 repetition 1



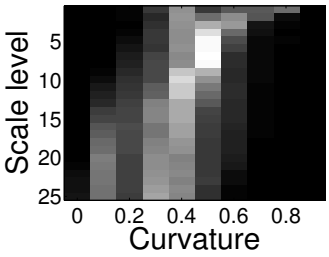
(e) G2 repetition 1



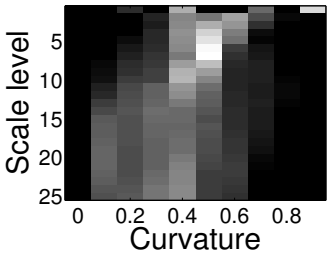
(f) G3 repetition 1



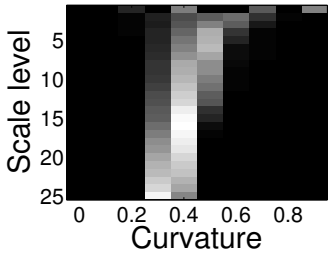
(g) G1 repetition 2



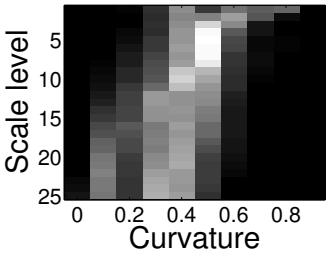
(h) G2 repetition 2



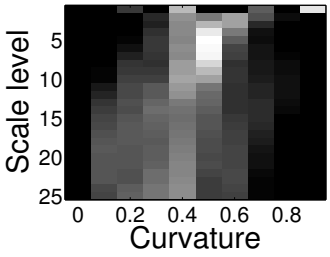
(i) G3 repetition 2



(j) G1 repetition 3



(k) G2 repetition 3



(l) G3 repetition 3

Figure 5.10: Comparison of the curvature descriptor for three different gestures

5.1.5 Palm morphology features

Palm morphology features are extracted from the point cloud of the palm samples in $\mathcal{P}$ in order to detect which fingers are likely to be folded over the palm, on the basis of the deformation the palm shape undergoes in the corresponding region when the related finger is folded or is raised [54, 56]. It is worth noting that the samples corresponding to the fingers folded over the palm belong to $\mathcal{P}$ and are thus not considered by feature sets describing the hand contour, but provide relevant information on the fingers opening status.

Moreover, the folded fingers may lead to failure the plane fitting on $\mathcal{P}$ (Alg. 4.3) since the assumption of flatness of $\mathcal{P}$ is no longer valid. Palm detection, in facts, has no preliminary information on possible folded fingers before fitting the plane and can not exclude the finger samples from the computation.

The main idea of this descriptor consists in detecting the palm regions where each finger can fold and compute for each of them the average distances from the palm plane of the samples within the region. The rationale is that the average sample distance for regions not occupied by a folded finger tends to 0, since the points almost lie on the palm plane, while for regions occupied by folded fingers the average distance is much higher as the samples are sensibly detached from the palm plane.

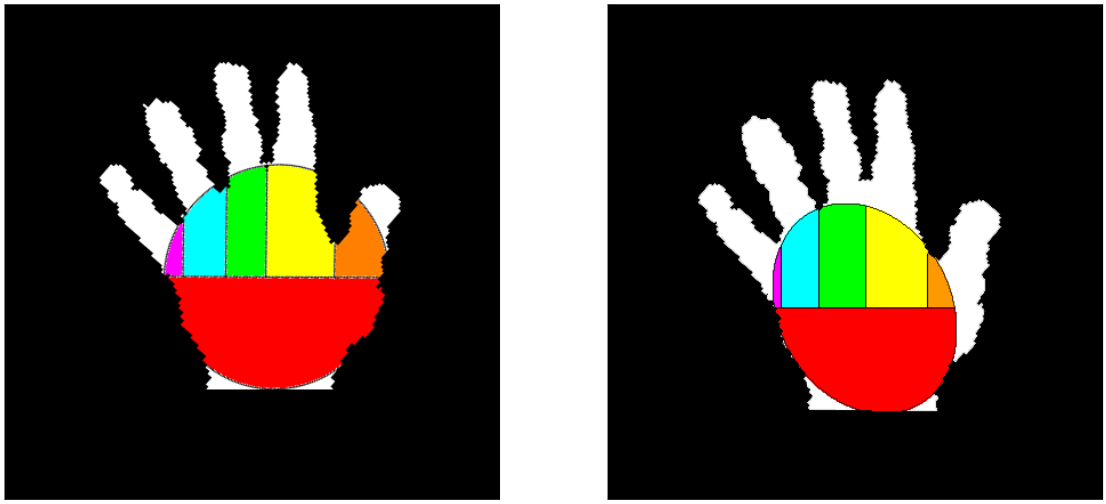
Let $\mathbf{u_p}$, $\mathbf{v_p}$ and c_p denote the hand local coordinate system on the range camera image plane computed in Section 4.2.2. The feature extraction procedure starts by partitioning $\mathcal{P}$ in six subsets: one subset ($\mathcal{P}_L$) only contains actual palm samples, located in the lower half of $\mathcal{P}$ (Eq. 5.21), and the remaining ones, denoted by $\mathcal{A}_j, j = 1, .., 5$ contain the samples of $\mathcal{P}$ which may either correspond to a folded finger or the upper palm region where the finger may fold.

$$\mathcal{P}_L = \{X_i \in \mathcal{P} | u_p > 0\} \tag{5.21}$$

with u_p the first coordinate value of the 3D point X_i projected in p_i on the range camera image plane and expressed in the palm coordinate system. An example of partitioning is shown in Fig. 5.11.

The proposed framework accounts for two possible partitioning of $\mathcal{P}$:

- **Fixed width intervals:** fingers are assumed to have the same uniform width, or equivalently the same angular extension in $L(\theta_q)$ (Eq. 5.4).
- **Variable width intervals:** fingers may have different widths or angular extensions in $L(\theta_q)$.



(a) Partition of a circular palm

(b) Partition of an elliptic palm

Figure 5.11: Example of palm region partitioning

Let $X_i \in \mathcal{P}$ denote a generic palm point in the 3D space and p_i its projection on the range camera image plane. Let also r_p be the palm radius estimated in Section 4.1. X_i is now assigned to a given finger $j = 1, \ldots, 5$ according to what range the value of its second coordinate in the hand 2D local coordinate system (v_p) p_i falls in. The assignment procedure is the same for both fixed or variable intervals, whose extensions are the only difference between the two cases. Fixed intervals have ranges $[-r_p, -{}^3/_5 r_p)$, $[-{}^3/_5 r_p, {}^1/_5 r_p]$, $[{}^1/_5 r_p, {}^3/_5 r_p)$ and $[{}^3/_5 r_p, r_p]$, while variable intervals have non-overlapping ranges dependent to the angular region assigned to each finger.

Recall that the angular interval of each finger is relative to the hand main direction which, besides being rather unstable, depends on the number and the direction of the raised fingers. Moreover, the angular intervals are only defined for the raised fingers, as they are clearly distinguishable in the finger contour outline $L(\theta_q)$. It follows that the actual angular interval for each finger is only disclosed in gestures having all the fingers raised (e.g., open hand).

For this reason, a preliminary step in the palm points finger region assignment consists in mapping the known finger intervals of the reference gesture g_o, denoting the open hand gesture (only used for the initial user calibration if not accounted in the gesture dictionary), to the currently performed gesture to analyze. The only needed information for this purpose is the circular shift ϕ_{g_o} between the distance plot $L(\theta_q)$ and the reference gesture g_o. Assuming to pin both the reference gesture and the current one on the same palm center, the alignment between the

two gestures is concluded by a rotation of ϕ_{g_o} of the gesture coordinate system (or equivalently a circular shift of $L(\theta_q)$ of ϕ_{g_o}). Finally, since the two gestures have different palm radiuses $r_p^{g_o}$ and r_p, a scaling factor $r_p/r_p^{g_o}$ is used to adapt the reference gesture finger intervals to the current gesture palm size.

$$v_p^{g_o} = \frac{r_p}{r_p^{g_o}}[-u_p \sin(\phi_{g_o}) + v_p \cos(\phi_{g_o})] \tag{5.22}$$

The success of the previously described gesture alignment strongly depends on the accuracy of the computed circular shift ϕ_{g_o}, which in turns depends on the employed plot similarity metric. As stated in Section 5.1.1, there are situations where the plot alignment fails, e.g. when two plots have similar outlines but different amplitude. In the particular case of the open hand gesture, it often happens that distance plots referred to gestures showing only one or two raised fingers are erroneously aligned to the reference plot for the open hand. In this case, the angular intervals are wrongly aligned with the performed gesture as well, hence the upper palm samples will not be assigned to the correct folded finger regions.

This problem is solved with a similar approach adopted for the distance descriptor of Section 5.1.1, namely by aligning the performed gesture with each gesture template and extract the same descriptor for each alignment. The main idea in this case consists in further aligning the mapped angular intervals with each gesture template and then extract the morphology descriptor after each alignment, since the finger regions in the gesture templates are known a priori and the alignment between each gesture template with g_o can be compensated during the calibration phase.

The adjusted angular sector mapping simply requires to replace ϕ_{g_o} with $\phi_{g_o} + \phi_{g_o,g}$ in Eq. 5.22, where $\phi_{g_o,g}$ denotes the circular shift between the open hand gesture template and the gesture g one. This expedient allows to compensate for the possible wrong alignments performed in Section 5.1.1.

Once the finger regions in the current gesture for the selected template alignment have been determined, Eq. 5.23 partitions the upper palm in five subsets $\mathcal{A}_{g,j}$ corresponding to the palm regions where a the j-th finger is folded or can fold.

$$\mathcal{A}_{g,j} = \{X_i \in \mathcal{P} \setminus \mathcal{P}_L | v_p \in \mathcal{I}_{g,j}^r\} \tag{5.23}$$

where $\mathcal{I}^r_{g,j}$ denotes the linear interval of v_p assigned to the j-th finger in the alignment with the g-th gesture template.

After assigning each point X_i to a region $A_{g,j}$, a new 3D plane π_L is fitted to the actual palm point set $\mathcal{P}_L$ with Alg. 4.3 as already done for the palm plane in Section 4.2.1. Then, for each generated subset $\mathcal{A}_{g,j}$ the feature extractor algorithm computes the average signed distance of its points from the plane π_L with Eq. 5.24 and scales it in the range $[-1, 1]$ by the the length L^{max} of the longest finger. Let $f^a_{g,j}$ denote this value.

$$f^a_{g,j} = \frac{\sum\limits_{X_i \in \mathcal{A}_{g,j}} \operatorname{sgn}\left((\mathbf{X_i} - \mathbf{X_i^\pi}) \cdot \mathbf{z_p}\right) \|\mathbf{X_i} - \mathbf{X_i^\pi}\|}{|\mathcal{A}_{g,j}|} \tag{5.24}$$

All the area features are collected within vector $\mathbf{F}^a$, made by $G \times 5$ area features, one for each finger in each gesture template.

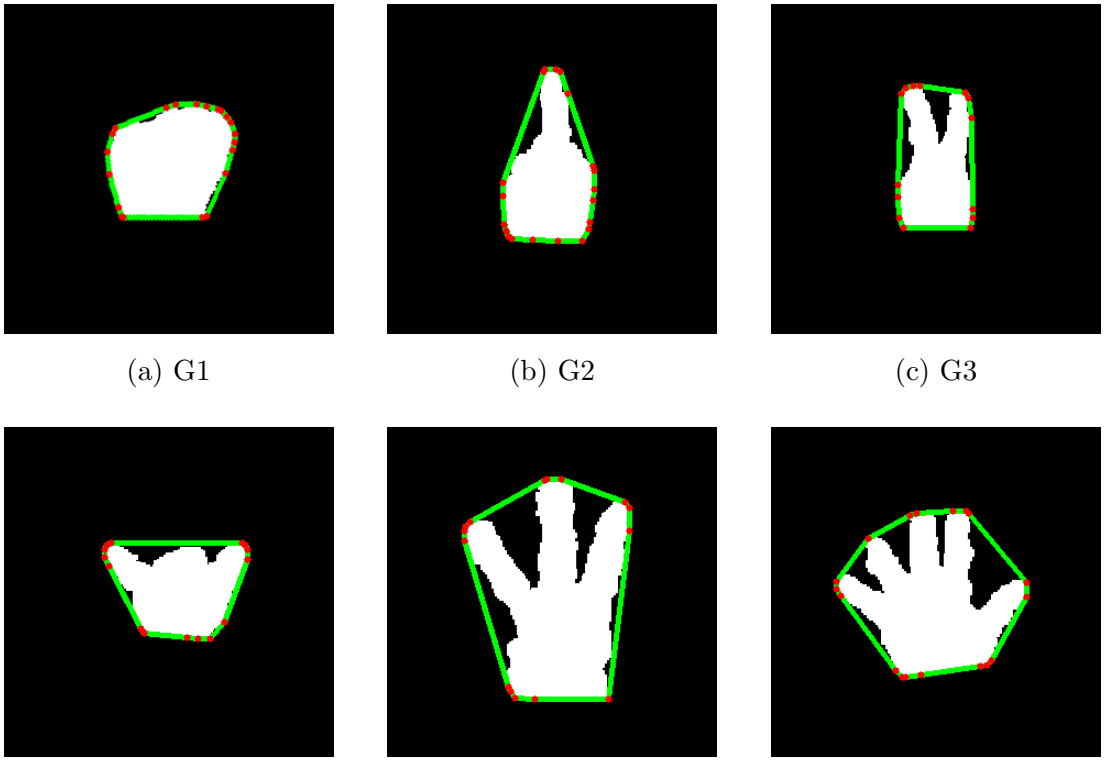
5.1.6 Convex hull features

The convex hull of the hand shape in the acquired depth map is another useful clue proposed in [19] and used in various gesture recognition schemes in literature.

Let B_A denote again the binary mask of the hand region after the wrist removal of Section 4.3 and $\mathcal{B}_A$ the set of coordinates of the pixels selected by B_A. The convex hull of the hand shape, denoted as $\mathcal{C}_H(\mathcal{B}_A)$, is by definition the smallest convex set containing $\mathcal{B}_A$, as exemplified in Fig. 5.12.

The main idea behind [19] and other schemes consists in computing the convex hull of the hand shape in the depth map and analyze the "convexity defects", namely the differences between the hand contour and the convex hull outline due to the empty spaces between finger pairs. These regions, in facts, strongly characterize each performed gesture. Fig. 5.12, for example, shows that the fist (G1) has almost no empty space within its convex hull, while the convex hull of G5 contains a significant amount of empty space. It follows that the fist will be hardly misrecognized as another sign with one or more raised fingers if an analysis on the convex hull empty space is performed.

The convex hull descriptors implemented in this work are based on the previous rationale and are extracted from the depth mask B_A. Due to the measurement noise in the depth samples, the hand shape in B_A and the associated convex hull are affected by several problems which, if not solved, lead to incorrect assumption in the further recognition step:



(a) G1 (b) G2 (c) G3

(d) G4 (e) G5 (f) G6

Figure 5.12: Example of extracted convex hulls from a few hand shapes

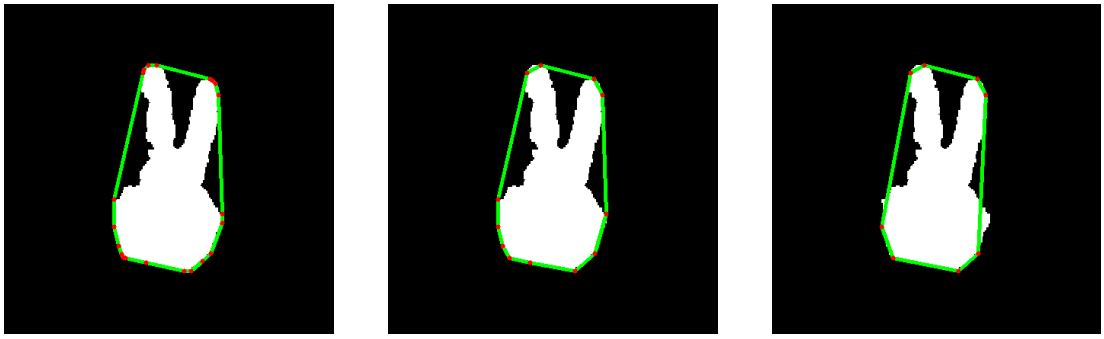
- B_A contains some small holes due to noise or missing data.
- The computed convex hull may have several close vertexes (highlighted in red in Fig. 5.12) due to the irregular shape of the acquired hand contour and consequently a considerable number of short edges.
- The presence of angles close to 180° between consequent convex hull edges (e.g. the edges almost lie on the same line) is a sign of extra edges due to acquisition artifacts.

The first problem is easily solved by state-of-art blob analysis techniques followed by a hole filling method. The second and third problems require, instead, the application of an ad-hoc convex hull simplification procedure able to remove the unnecessary vertexes without sensibly reducing the convex hull accuracy.

The convex hull simplification procedure implemented in this work and formalized in Alg. 5.1 consists in firstly collapsing all the vertexes connected by edges whose length is lower than a given threshold T_L^{CH} until every edge of the simplified convex hull is sufficiently long, and then in removing all the vertexes

$v_i, i = 1, \ldots, N$ whose angle $\angle v_{(i-1) \mod N} v_i v_{(i+1) \mod N}$ value is higher than an angular threshold T_θ^{CH}. For the experiments of Chapter7 T_L^{CH} and T_θ^{CH} where set respectively to $10mm$ and 160°. Note how T_L^{CH} is adapted in [pxl] by Eq. 5.16 replacing R_s with T_L^{CH}.

The resulting simplified convex hull $\mathcal{C}_H(\mathcal{B}_A)$ is the starting point for the extraction of the various features described in the following subsections.



(a) Original convex hull

(b) Convex hull after short edges removal

(c) Convex hull after wide angle vertexes removal

Figure 5.13: Example of convex hull simplification

Convex hull vertexes number

A first possible feature is the number of vertexes of the simplified convex hull [54]. This value is an hint of the hand pose and in particular of the number of raised fingers, since the ideal convex hull has a vertex for each fingertip and a few other vertexes delimiting the palm area.

While this is theoretically correct, the measurement noise in the depth values makes this descriptor unusable in practical situation. The number of convex hull vertexes for repetitions of the same gesture is, in fact, highly unstable.

Perimeters ratio

The ratio between the perimeter of the hand contour in B_A and the perimeter of the convex hull is another useful clue [54]. Gestures with folded fingers typically report perimeter ratios close to 1 (e.g., G1 in Fig. 5.12), while the perimeter ratio for gestures with several raised fingers is usually smaller.

$$F_p^{ch} = \frac{|\partial B_A|}{perimeter(\mathcal{C}_H(\mathcal{B}_A))} \tag{5.25}$$

Algorithm 5.1 Convex hull simplification algorithm

Input:

$\mathcal{V}$: ordered list of the original convex hull vertexes $v_i, i = 1, \ldots, N$

T_L^{CH}: minimum allowed distance between consequent vertex pairs

T_θ^{CH}: maximum allowed angle between consequent edge pairs

Output: $\mathcal{S}$: ordered list of the simplified convex hull vertexes $v_j^S, j = 1, \ldots, M \leq N$

$curr \leftarrow 0, prev \leftarrow N - 1$ ▷ Current and previous vertex indexes

$\mathcal{S} \leftarrow \{v_{prev}\}$

repeat

$prevvert \leftarrow v_{prev}, currvert \leftarrow v_{curr}$ ▷ Current and previous vertexes

$dist \leftarrow \|\mathbf{currvert} - \mathbf{prevvert}\|$

if $dist \geq T_L^{CH}$ **then**

$\mathcal{S} \leftarrow \mathcal{S} \cup \{currvert\}$

$prevvert \leftarrow currvert$

$prev \leftarrow (prev + 1) \mod N$

end if

$curr \leftarrow (curr + 1) \mod N$

until $curr \leq N$

repeat

$rempoint \leftarrow false$ ▷ Flag indicating whether at least one vertex has been removed

$angles \leftarrow array(\theta)$ of $|\mathcal{S}|$ elements

for $i \leftarrow 0, 1, \ldots, |\mathcal{S}|$ **do**

$prevvert \leftarrow v^S_{(i-1) \mod |\mathcal{S}|}, currvert \leftarrow v_i^S, nextvert \leftarrow v^S_{(i+1) \mod |\mathcal{S}|}$

$angles[i] \leftarrow \angle prevvert, currvert, nextvert$

end for

$(maxangle, idx) \leftarrow \text{MAX}(angles)$

if $maxangle > T_\theta^{CH}$ **then**

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{v_{idx}^S\}$

$rempoints \leftarrow true$

else

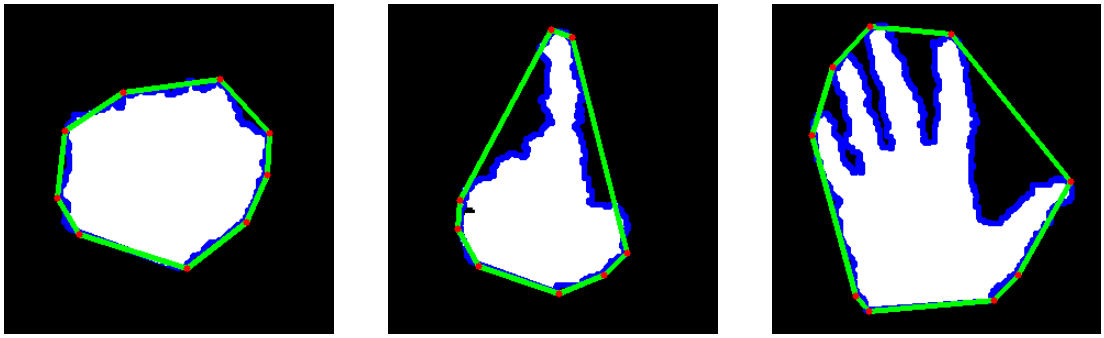
$rempoints \leftarrow false$

end if

until $rempoints = true$

return $\mathcal{S}$

Fig. 5.14 shows a few examples of perimeter ratios.



(a) G1 (b) G2 (c) G3

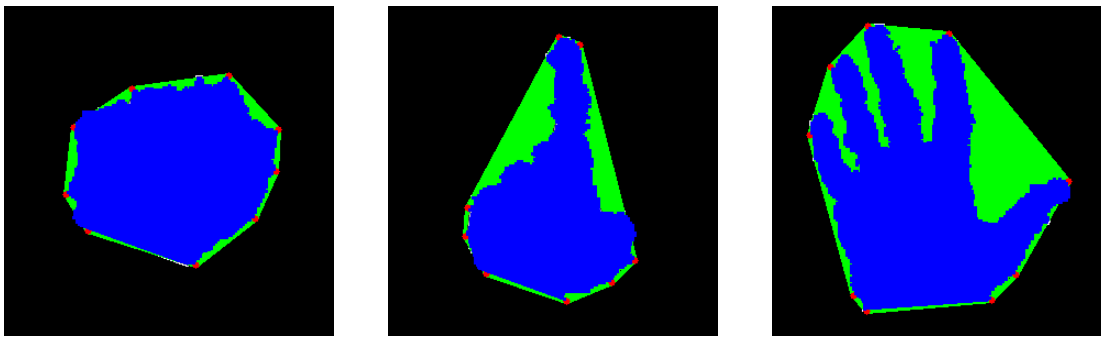
Figure 5.14: Comparison of hand VS convex hull perimeter ratios

Finally, since the simplified convex hull edges may intersect the hand region, the hand samples outside the convex hull area are previously discarded and not accounted in Eq. 5.25.

Areas ratio

The ratio between the area of the hand shape and the area of the associated convex hull offers a similar description of the one provided by Eq. 5.25 [54].

$$F_a^{ch} = \frac{|\mathcal{B}_A|}{area(\mathcal{C}_H(\mathcal{B}_A))} \tag{5.26}$$



(a) G1 (b) G2 (c) G3

Figure 5.15: Comparison of hand VS convex hull area ratios

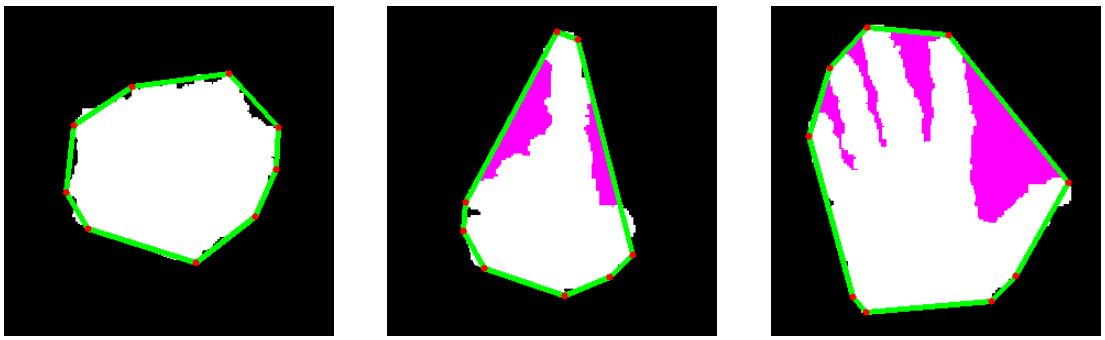
Note how, while the perimeter value of the hand shape is always higher than the convex hull one, for the area ratio this property is reversed.

Connected components features

One of the other relevant cues that can be extracted from the comparison between the convex hull and the hand region comes from the analysis of the empty space in the convex hull not occupied by the hand samples [57, 54].

Let B_{CH} denote the binary mask representing the convex hull region in the range camera image plane, and by $S = B_{CH} - B_A$ the difference between the convex hull and the hand regions. S is typically made of a set of connected components (blobs) S_i of various size, as exemplified in Fig. 5.16. Note how G1 is characterized by the lack of connected components while G2 and G3 by two or more components of significant size.

The extraction algorithm for the current descriptor firstly performs blob analysis on S to only retain the connected components whose area is higher than a preset threshold T_a^{cc}. This selection is necessary to avoid considering small components due to noise. The set $\mathcal{S} = \{S_i | S_i > T_a^{cc}\}$ of the retained connected components is at the basis of all the features described in this section.



(a) G1 (b) G2 (c) G3

Figure 5.16: Comparison of convex hull connected components

A first feature that can be extracted is the number of the retained connected components $N_{cc} = |\mathcal{S}|$. Another feature set is, instead, given by the ratio of the areas of the various connected components over the convex hull area, that is:

$$A_i^{cc} = \frac{area(S_i \in \mathcal{S})}{area(\mathcal{C}_H(\mathcal{B}_A))} \tag{5.27}$$

It is worth noting that the number and the area ratios of the retained connected components are not constant and depend not only by the class of the performed gesture, but also by the different way the same gesture is repeated over time. Moreover, in order to compare the connected components area ratios among

the various gestures, it is necessary to impose an ordering among the features A_i^{cc}. For this reasons, the retained connected components are firstly sorted in descending order respect to their blob areas, then only the first $N_{cc} = \min(|\mathcal{S}|, N_{max})$ of them with the highest areas are kept, where N_{max} is the maximum number of desired components (for the tests of Chapter 7, $N_{cc} = 6$). Finally, the N_{cc} retained connected components are sorted again in ascending order respect to the angle θ_j formed by the segment joining the component center s_j (blob centroid) with the palm centroid c_p in B_A, and the main axis of the hand coordinate system defined in Section 4.2.2. The procedure is exemplified graphically in Fig. 5.17.

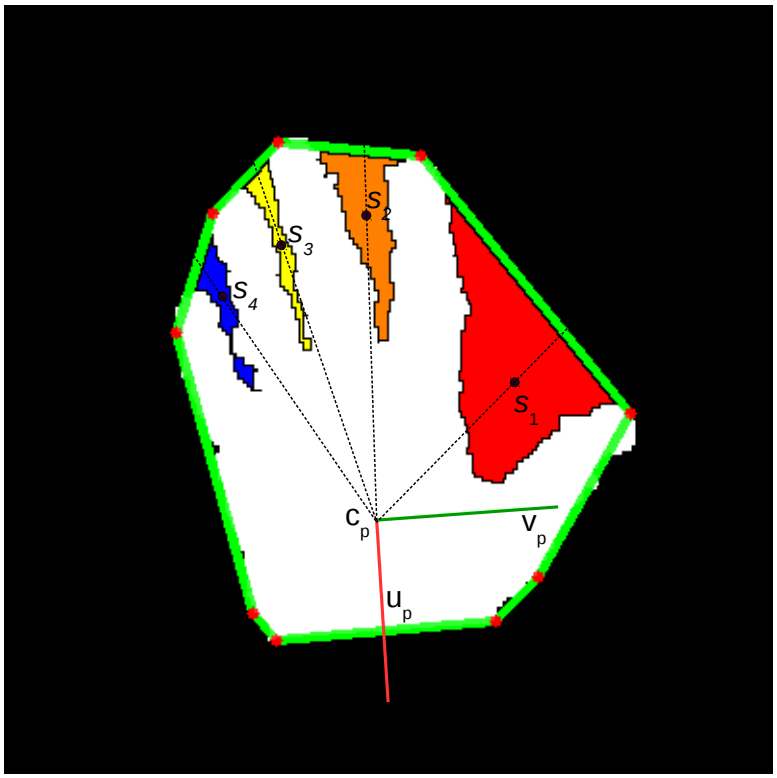


Figure 5.17: Connected components sorting within the computed convex hull

Let $j = 1, \ldots, N_{cc}$ be the index of the retained and ordered connected components resulting from the previous processing. The connected components feature vector $\mathbf{F_{cc}^{ch}}$ is built according to Eq. 5.28.

$$f_j^{cc} = \mathcal{A}_j^{cc} \tag{5.28}$$

Note how in case $N_{cc} < N_{max}$, the feature vector is padded with 0.

5.1.7 Fingertip orientations

Fingertip orientations feature set describe the angle formed by the segments joining each fingertip projection on the palm plane with the palm centroid and the hand direction. The computation of this feature set plays a key role not only as a stand-alone descriptor, but also for the extraction of other descriptors since the angle is used as a metric to order the fingertips.

Let F_i for $i = 1, \ldots, 5$ be the i-th fingertip and θ_i the angle formed with the hand direction (Eq. 5.29).

$$\theta_i = \angle(\mathbf{F}_i^{\pi} - \mathbf{C_p}, \mathbf{x_p}) \tag{5.29}$$

where F_i^{π} denotes the projection of F_i on the palm plane π.

Fingertip F_i detection, a preliminary step required for the extraction of the current descriptor, is a rather hard problem to solve even with the aid of depth data. One possible solution, proposed in this work, consists in exploiting the distance plot $L(\theta_q)$ generated in Section 5.1.1 for the hand contour distance descriptor due to its capability of describing accurately the hand shape. The rationale behind this solution is that $L(\theta_q)$ encodes the searched fingertip positions, as for each angular position in $L(\theta_q)$ it is also stored a reference to the associated hand point X_i. It follows that the global maxima of $L(\theta_q)$ are likely to be associated to the fingertips, as fingertips are usually the most distant points from the palm centroid C_P.

Note how, due to the noise which may not have been removed by the Gaussian filtering in Section 5.1.1, a simple peak detection based on the evaluation of the first and second order derivatives of $L(\theta_q)$ may return spurious global maxima along with the actual fingertips (Fig. 5.18). Although most of the unnecessary peaks can be removed by simple considerations on their value or position (e.g., peaks lower than a given fraction of the maximum finger length L^{max} or lying in the neighborhood of another higher peak are more likely to be due to noise), some of them may still be retained. In the worst case, e.g. adjacent fingers touching, some fingertips may also be undetected as they are not maxima in $L(\theta_q)$ (lost peaks) or because the finger is folded over the palm.

Let P_i for $i = 1, \ldots, N \leq 5$ denote one of the first N peaks detected on the considered $L(\theta_q)$ and retained by the previous rationale. Note how the maximum accepted peak number is 5 as they are likely to correspond to the 5 different fingertips. Assume again to sort the 3D points associated to the N peaks according to the angles they form with the hand direction, as done in Section 5.1.6 for the

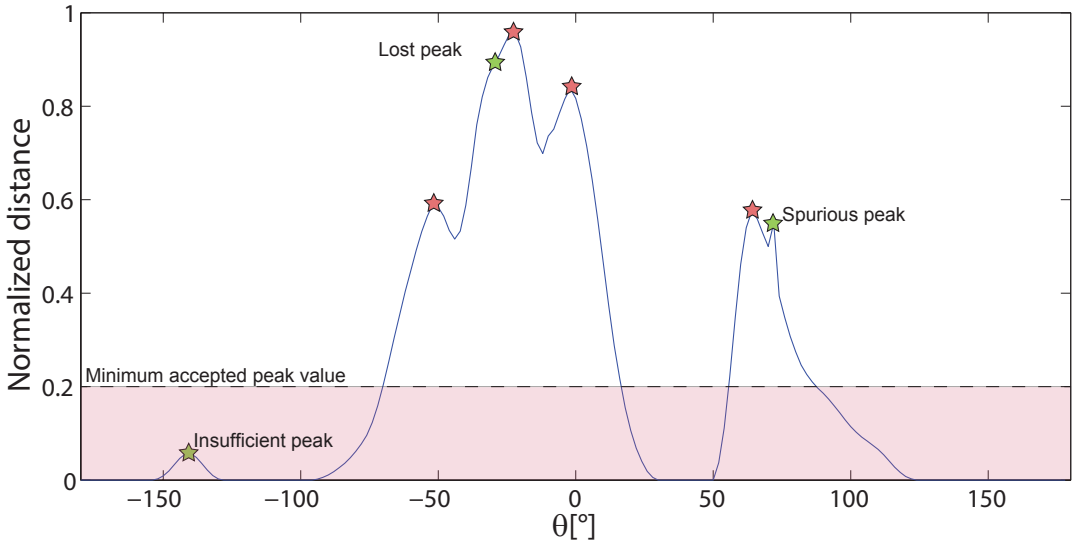


Figure 5.18: Example of detected maxima on $L(\theta_q)$

convex hull connected components. Let P_j for $j = 1, \ldots, N \leq 5$ denote the sorted peaks.

The current descriptor is made by the juxtaposition of the sorted fingertip angles θ_j in the feature vector $\mathbf{F}^\theta$, scaled in the interval $[0.5, 1]$, padding the possible missing values with 0. Note that the interval starts from 0.5 and not from -1, although $[-1, 1]$ is the most common interval for features also assuming negative values, to better discriminate the case of $\theta_j = 0$ from the 0 padding of the missing values.

5.1.8 Fingertip positions

The coordinates of the fingertips F_j in the palm coordinate system, extracted for the construction of the descriptor of Section 5.1.7, allow the realization of another powerful descriptor encoding in a more compact way the information provided by the fingertip maximum distances from the palm center C_P and from the palm plane π, and their orientations θ_j respect to the hand main direction.

This descriptor is built by the same algorithm of Section 5.1.7 employed for the realization of the fingertip angles vector $\mathbf{F}^\theta$, replacing each fingertip scaled angle in $\mathbf{F}^\theta$ with the associated 3D point X_i coordinates in the hand local reference system. Each fingertip coordinate is scaled into the same range $[0.5, 1]$ by the length L^{max} of the longest user's finger.

Note how the new feature vector $\mathbf{F^P}$ has 15 entries instead of the 5 of $\mathbf{F}^\theta$, as each fingertip is now described by 3 values, one per coordinate.

5.2 Leap Motion features

The Leap Motion, described previously in Section 2.8, is a novel low-cost device designed for gesture recognition purposes only which is raising an high interest thanks to the information about the hand posture that is able to return in real-time. Recall that, differently from the range cameras described in Chapter 2, the Leap Motion does not return a depth map of the framed scene but only a set of relevant hand key points and some hand pose features (Fig. 2.20).

The estimated 3D hand center and orientation, for example, have a fundamental importance for the fast hand tracking and hand detection, heavy computational tasks that are performed by most of the methods of Section 1.2 by exploiting depth, color or other types of information. The estimated fingertip coordinates are, instead, employed in the extraction of the descriptors of Sections 5.1.7 and 5.1.8, resulting from a complex processing of the acquired depth map and the adoption of a few anthropometric considerations.

An important observation is that, while the computed 3D fingertip positions are quite accurate (the error is about 200 μm according to the study in [45]), the sensor is not always able to recognize all the fingers. Fingers touching each other, folded over the hand or hidden from the camera viewpoint are naturally not captured, but also in several configurations some visible fingers could be lost, specially if the hand is not perpendicular to the camera. Moreover, protruding objects near the hand, like bracelets or sleeve edges, are easily misrecognized as fingers. This is quite critical, since in different executions of the same gesture the number of captured fingers could vary. Note also that the first release of the Leap Motion software does not return any information about the matching between the acquired points and the corresponding fingers, hence the estimated fingertip positions are returned in random order.

Finally, the estimated hand center, differently from the one of Section 4.3, is highly unstable and does not necessarily correspond to the actual palm center, but moves according to the fingers configuration in the performed gesture.

The remaining of current section presents several descriptors that can be extracted from the Leap Motion data and are considered in the proposed framework:

- **Fingertip orientations:** angles corresponding to the orientation of each fingertip projected on the palm plane (defined by the palm normal $\mathbf{n}$ and centered on the hand center C_L) with respect to the hand orientation $\mathbf{h}$.
- **Fingertip positions:** x, y and z coordinates of the detected fingertips in the hand coordinate system defined by the directions $\mathbf{h}$, $\mathbf{n}$ and C_L.

- **Fingertip distances from the palm center:** distances of the fingertips from the hand center C_L.
- **Fingertip distances from the palm plane:** distances of the fingertips from the palm plane defined by the palm normal $\mathbf{n}$ and centered on the hand center C_L.
- **Inter fingertip distances:** distances between consecutive fingertips.
- **Inter fingertip orientations:** angles formed by the segments joining pairs of consecutive fingertips with the hand center C_L.
- **Hand radius:** the radius of the sphere which best approximates the palm surface.
- **Number of fingertips:** the number of the detected actual fingertips.

All the feature values are normalized in the interval $[0, 1]$ or $[05, 1]$ by dividing the values for the length L^{max} of the longest finger (e.g., middle finger) in order to make the approach robust to people with hands of different size. The scale factor L^{max} can be computed during the system calibration.

5.2.1 Fingertip orientations

This descriptor is analogous to the one defined in Section 5.1.7, but has a simpler construction [57, 58]. While, in facts, the feature extraction algorithm of Section 5.1.7 requires a prior fingertip estimation exploiting the distance plots of Section 5.1.1, the use of a Leap Motion as acquisition device allows to skip this step as the fingertip positions in 3D space are already provided by the sensor APIs.

Recall that the returned 3D points are not necessarily all associated to actual fingertips but some of them could be due to noise in the measurement. For each fingertip candidate P_i, its angle respect to $\mathbf{h}$ (Eq. 5.29) is used both for imposing an ordering on the fingertips, as they are normally returned by the Leap Motion in random order, and to discriminate the actual fingertips from spurious ones detected on sleeves. Fingertip angle values outside range $[-90°, 90°)$ are, in facts, due to artifacts, e.g. crests in the hand arm sleeve. Such fingertip candidates are, thus, discarded and will not contribute to this descriptor.

The retained fingertip candidates P_j with $j = 1, \ldots, N_F \leq 5$ are then sorted in ascending order respect to the angle formed with $\mathbf{h}$ and normalized in the range $[0.5, 1]$ by the maximum angle from $\mathbf{h}$, namely $\theta^{max} = 90°$. The eventual missing fingertip values are again padded with 0.

Fingertip orientations, collected in a feature vector $\mathbf{F}_{\mathbf{L}}^{\theta}$ provide, again, an important information both to be used alone as descriptor, or better as the basis for the construction of more informative feature sets.

5.2.2 Fingertip distances from the palm center

This feature set reports the fingertip distances from the palm center, similarly to the depth descriptor of Section 5.1.1 but, this time, only accounting for the distances of the actual fingertips and not of every generic hand contour point [57, 58].

Let C_L denote the hand center returned by the Leap Motion and $F_j, j = 1, \ldots, N_F \leq 5$ a generic fingertip retained and sorted by the procedure described in Sections 5.2.1. The distance D_j of F_j from C_L is defined as:

$$D_j = \frac{||\mathbf{F}_j - \mathbf{C_L}||}{L^{max}} \tag{5.30}$$

This scaling is needed to normalize all distances to the range $[0, 1]$ to account for the different hand sizes of the various users. Although the minimum distance is 0, the scaled distances are then shifted to the range $[0.5, 1]$ to discriminate a missing fingertip from a fingertip distance close to 0.

Note how, differently from the distance descriptor of Section 5.1.1, this time there are no gesture templates and, although the fingertips are sorted according to the angle respect to the hand direction $\mathbf{h}$, distances themselves are not sufficient to discriminate all the gestures. Differences among gestures characterized by the same number of raised fingers with a similar arrangement (e.g. the gestures in Fig. 5.19) are not always captured by this descriptor.

(a) First gesture

(b) Second gesture

Figure 5.19: Example of gestures not discriminable by the Leap Motion

In this work, the problem is tackled by partitioning the plane defined by $\mathbf{n}$ and passing through $\mathbf{C_L}$ into five angular regions S_i, $i = 1, \ldots, 5$ (Fig. 5.20), and assign each captured finger F_j to a specific region according to the angle θ_j between the projection of the finger in the plane and the hand direction $\mathbf{h}$ (Eq. 5.29).

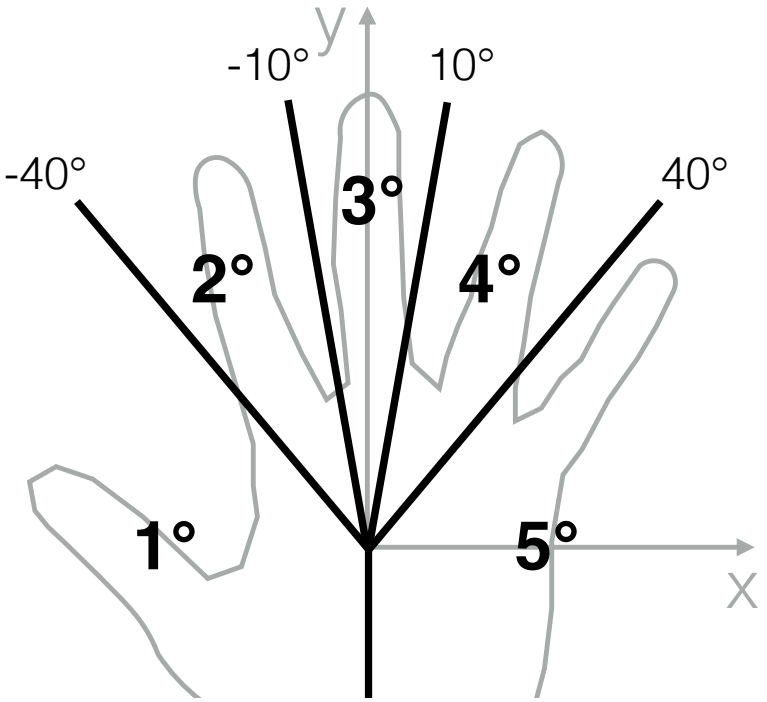


Figure 5.20: Angular regions in the palm plane.

It is worth noting that there is not a one-to-one matching between sectors and fingers, namely, some of the sectors S_i could contain more than one finger and others could be empty. When two fingers compete for the same sector S_i, one of them is assigned to the nearest adjacent sector S_{i+1} or S_{i-1} if not already occupied, otherwise the maximum between the two feature values competing for S_i is selected. Let us assume sector S_{i-1} is empty and let denote by F_i, F_j respectively the fingertip assigned to S_i and the one to be assigned to a sector. F_j will be assigned to S_{i-1} if $\theta_j \leq \theta_i$, with θ_j and θ_i denoting the orientation values of F_j and F_i, otherwise F_i will be reassigned to S_{i-1} and F_j to S_i. The same rationale holds in case S_{i+1} is empty.

The current descriptor, represented by a feature vector $\mathbf{F}_L^d$ consists, then, in the juxtaposition of the fingertip distances after the assignment to the sectors S_i, setting a 0 value only for the possible empty positions.

5.2.3 Fingertip distances from the palm plane

This descriptor follows the same rationale and construction of the one of Section 5.2.2, replacing the distance of the fingertips from the hand center with their distances from the plane defined by the palm normal $\mathbf{n}$ returned by the Leap Motion [57, 58] (Eq. 5.31).

$$E_j = \text{sgn}\left((\mathbf{F_j} - \mathbf{F_j^n}) \cdot \mathbf{n}\right) \frac{\|\mathbf{F_j} - \mathbf{F_j^n}\|}{L^{max}} \tag{5.31}$$

where $\mathbf{F_j}^{\mathbf{n}}$ denotes the projection of $\mathbf{F_j}$ on the plane defined by $\mathbf{n}$.

Analogously to the similar descriptor of Section 5.1.2, the sign operator in Eq. 5.31 discriminates to which of the two semi-spaces defined by the palm plane the fingertip belongs. The values $E_j, j = 1, \ldots, N_F \leq 5$ are then assigned to different angular sectors according to the rationale of Section 5.2.2. Again, there is at most one feature value for each sector and the missing values are set to 0. The values range of this descriptor can assume both positive and negative values, but are scaled to the interval $[0.5, 1]$. All the E_j are collected into a vector $\mathbf{F}_L^e$.

5.2.4 Fingertip positions

This feature set represents the positions of the fingertips in the 3D space estimated by the Leap Motion software [57]. As for the analogous descriptor of Section 5.1.8, since a reliable hand gesture recognition system must be independent from the hand position and orientation, the fingertip coordinates are expressed in the Leap Motion reference system according to Eq. 5.32.

$$\begin{aligned} F_j^x &= (\mathbf{P}_j - \mathbf{C_L}) \cdot (\mathbf{n} \times \mathbf{h}) \\ F_j^y &= (\mathbf{P}_j - \mathbf{C_L}) \cdot \mathbf{h} \\ F_j^z &= (\mathbf{P}_j - \mathbf{C_L}) \cdot \mathbf{n} \end{aligned} \tag{5.32}$$

where $P_j, j = 1, \ldots, N_F \leq 5$ are the retained and sorted fingertip candidates according to the angle criteria of Section 5.2.1.

The computed fingertip coordinates are normalized in the range $[0.5, 1]$ using the longest finger length L^{max} as scale factor. The possible missing fingertip values are again padded with 0. Finally, it is worth noting that the fingertip 3D positions can be seen, as for the analogue descriptor of Section 5.1.8, as the compact representation of the combination of angle, distance and elevation information. Fingertip positions are collected into a feature vector $\mathbf{F_L^p}$.

5.2.5 Inter fingertip distances

Fingertip distances from the hand center, described in Section 5.2.2, are a powerful yet imperfect descriptor that in a few cases is not able to reliably discriminate gestures with the same number of fingers with similar lengths (Fig. 5.19). This is due to the fact that such descriptor does not account for the absolute position of the fingertips within the gesture.

A possible solution, implemented in this work and described in Section 5.2.2, consists in partitioning the palm plane in sectors where each fingertip projection is likely to "fall into", where each sector is associated to a known finger (e.g., a fingertip falling in the rightmost sector in Fig. 5.20 is likely to belong to the actual pinky finger). The drawbacks are the need for defining fixed angular sectors on the open hand reference gesture and, since the sectors are defined respect to the hand direction which varies among different gestures, the sectors may not always correspond to the same fingers.

An efficient and alternative solution to overcome this ambiguity, implemented in the proposed framework, consists in augmenting the distance descriptor of Section 5.2.2 with information about the relative distance between adjacent fingertips (in the ordering) and avoiding the fingertip assignment to separate sectors. The rationale is that the inter-distance between adjacent fingertips may help to discriminate gestures sharing the same number of fingertips and their distances from the hand center but a different arrangement.

Let $I_j, j = 1, \ldots, 4$ denote the relative Euclidean distance in 3D space between fingertip F_j and the adjacent one F_{j+1}, as reported in Eq. 5.33. Note how the fingertips F_j come from the fingertip candidate processing using the approach described in Section 5.2.1.

$$I_j = \frac{\|\mathbf{F_{j+1}} - \mathbf{F_j}\|}{I^{max}} \tag{5.33}$$

where I^{max} is a scaling factor set with the highest distance between fingertips, e.g., the distance between the thumb and the pinky. As there may be up to 4 pairs of adjacent fingers and the missing values are padded with 0, the range for these features is set again to $[0.5, 1]$. The fingertip inter distances are collected into a feature vector $\mathbf{F_L^i}$.

5.2.6 Inter fingertip orientations

This feature set is an alternative to the fingertip inter-distance descriptor of Section 5.2.5, based on replacing the inter finger distances with the angles θ_j^f formed by the segments joining the projections $F_j^{\mathbf{n}}$ of adjacent fingertips on the palm plane (defined by $\mathbf{n}$) with the estimated hand center C_L.

Let F_j and F_{j+1} denote, again, two retained and sorted adjacent fingertips according to the rationale of Section 5.2.1. The angle $\angle F_j^{\mathbf{n}} C_L F_{j+1}^{\mathbf{n}}$ is computed by Eq. 5.34, based on a variation of Eq. 5.29.

$$\theta_j^f = \arccos\left(\frac{\mathbf{F_{j+1}^n} - \mathbf{C_L}}{\|\mathbf{F_{j+1}^n} - \mathbf{C_L}\|} \cdot \frac{\mathbf{F_j^n} - \mathbf{C_L}}{\|\mathbf{F_j^n} - \mathbf{C_L}\|}\right), j = 1, \ldots, 4 \tag{5.34}$$

As for the descriptor of Section 5.2.5, there may be up to 4 pairs of adjacent fingers and the missing values are padded with 0. The range for these features is set to $[0.5, 1]$. The fingertip inter angles are collected into a feature vector $\mathbf{F_L^{\theta,int}}$.

5.2.7 Hand radius

Hand radius is another hand relevant information returned by Leap Motion APIs that, although if considered alone does not allow a reliable gesture recognition, when used along with other features may improve the overall recognition accuracy.

This feature is made by a single value $F_L^r = r/S_R$ that represents the scaled radius of a sphere that roughly approximates the hand palm surface. The scale factor S_R corresponds to the maximum sphere radius detected by the sensor.

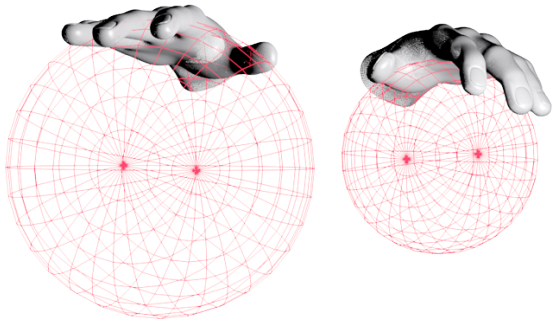


Figure 5.21: Example of hand radius detected by the Leap Motion

The rationale behind this feature consists in the fact that the sphere radius length is in a direct proportion with the overall finger opening status, e.g., bigger spheres are associated to open hand gestures, while smaller ones are associated

to gestures characterized by fingers curling into a fist. For these reasons, sphere radius is useful to state if the hand is open or the fingers are folding.

5.2.8 Number of detected fingers

The number of detected fingers is another feature directly defined on a value returned by the Leap Motion APIs.

This value simply reports the number of captured fingers N_F returned by the sensor. Recall that the value itself is not reliable, especially in presence of fingers touching each other or long sleeves, although the fingertip pre-processing described in Section 5.2.1 is able to compensate the Leap Motion fingertip detection errors in most cases.

The current feature is defined as $F_L^{\mathbf{n}} = N_F/5$ to constrain its values in the range $[0, 1]$.

5.3 Depth data features with Leap Motion aid

Several features described in Section 5.1 are defined on the local hand coordinate system, obtained as the result of a delicate and computationally expensive processing of the acquired depth map. Moreover, the hand and the palm orientations require further assumptions to state whether their directions are correct or must be inverted.

Recall that the Leap Motion APIs return a limited set of the same key points including the estimated hand center, the hand orientation and the fingertip positions in the camera space. In particular, Section 3.3 showed how the estimated hand center by the Leap Motion software may allow to reliably and quickly segment the hand from the background in the acquired depth map, relaxing the assumption of the hand being the nearest object to the range camera without resorting on the color information.

Following the previous rationale, this work also accounts for the possibility of using an hybrid acquisition setup made by a range camera and the Leap Motion (Section 2.9) in order to leverage the joint usage of depth information and the key points provided by the Leap Motion APIs for the extraction of the same features of Section 5.1.

When using the acquisition rig of Fig. 2.21, the hand detection, segmentation and the geometric feature extraction steps in the recognition pipeline of Fig. 1.8 can be simplified as follows.

- **Hand detection:** the hand center C_L estimated by the Leap Motion may be used in place of finding the nearest hand sample $X_{u,v}^{min}$ from the range camera in Eq. 3.3, without requiring a previous skin color thresholding or assuming the hand is the nearest object to the sensor.

- **Hand segmentation:** the hand center C_L, hand direction $\mathbf{h}$ and palm normal $\mathbf{n}$ returned by the Leap Motion APIs may avoid the need for the usage of PCA and plane fitting with RANSAC for the definition of the hand local coordinate system. Moreover, C_L may be used as the starting point for the palm detection of Alg. 4.2.

- **Feature extraction:** the fingertip coordinates estimated by the Leap Motion may be, after the spurious finger removal, used directly for the extraction of the fingertip orientation and position features, without requiring the distance plot analysis of Section 5.1.7.

Since the range camera and the Leap Motion have different reference systems, the acquisition setup requires an accurate calibration in order to jointly use the data from the two sensors. Differently from the color and depth cameras, though, the Leap Motion does not return a color or infrared image of the acquired scene, and the calibration protocols employed for the color and depth cameras ([1]) can then not be adapted to the Leap Motion case.

For this reason, this thesis proposes in the following section an ad-hoc calibration protocol designed for the jointly usage of a range camera and the Leap Motion in an hybrid setup. Note how the calibration must be repeated whenever the range camera or the Leap Motion are moved from their original positions, as the sensor alignment is lost.

5.3.1 Acquisition setup calibration

The hybrid setup calibration protocol aims at estimating the extrinsic parameters of the two devices, namely the roto-translation $(R, \mathbf{t})$ allowing to express the 3D points in a sensor reference system respect to the one of the other device and vice-versa. In addition, the two devices need also to be previously independently calibrated in order to correctly locate points in the 3D space. The Leap Motion software already provides a calibration tool, while range cameras require an ad-hoc approach like the one of [42] for Microsoft Kinect.

Assuming the range camera modellable by the pin-hole camera model (Section 2.1), its calibration only requires the estimation of the intrinsic parameter matrix

K_D, since the depth map already contains the needed information for mapping the 3D points in the range camera plane.

In order to find the roto-translation between the two sensors, the standard procedure consists in aligning a point cloud of key points whose coordinates are expressed in the first sensor reference system, with the point cloud of the corresponding key points expressed in the reference system of the other sensor. The alignment method finds the roto-translation $(R, \mathbf{t})$ minimizing in the least-square sense the average Euclidean distance between a general key point in the first system and the associated transformed key point in the the second system.

Usually the calibration of setups made by multiple range or color cameras exploits as key points the corners extracted from a checkerboard with known checkers number and size. The Leap Motion, instead, as it does not return any image or depth map, can only rely on the estimated positions in the sensor space of the detected fingertips as key points. For this reason, the proposed calibration protocol employs the 3D fingertip positions of the open hand estimated by both the range camera and the Leap Motion as calibration cues [57]. The choice of the open hand is due both because this gesture maximizes the number of key points extracted per frame, and because the Leap Motion is less likely to detect spurious fingers on this gesture.

It is worth noting that not using external tools like checkerboards or other classic calibration devices is not a limit of the proposed approach but, indeed, it is a key requirement for a human-computer interaction system. The calibration, in facts, is performed automatically and only requires the user to acquire a few frames of the open hand. Moreover, the two devices do not have to be rigidly attached to a fixed structure, as whenever one of them is moved the system recalibration only requires the acquisition of a few frames of the user's open hand. As shown in Fig. 2.21, in order to be able to retrieve useful information from both the sensors the Leap Motion has to be put under the performed gesture, while the depth sensor has to be placed a little more forward, facing the user, as in a regular gesture recognition setup.

Let $\mathbf{h}$, $\mathbf{n}$ denote, again, respectively the hand direction and palm orientation estimated by the Leap Motion, and by C_L the estimated palm center. The first step of the proposed calibration procedure consists in analyzing and sorting the detected fingertips for each calibration frame, as done in Section 5.2.1, to obtain a set of $5 \times N$ points $\mathcal{X}_L = \{X^1_{L,1}, \ldots, X^1_{L,5}, \ldots, X^N_{L,1}, \ldots, X^N_{L,5}\}$ describing the actual fingertip positions in the Leap Motion coordinate system for the i-th calibration frame, with $i = 1, \ldots, N$. For the range camera, in-

stead, the fingertip positions estimation from the depth map is much more complex and exploits the method described in Section 5.1.7, which returns a set $\mathcal{X}_D = \{X^1_{D,1}, \ldots, X^1_{D,5}, \ldots, X^N_{D,1}, \ldots, X^N_{D,5}\}$ denoting the extracted fingertip positions from the depth map in the range camera reference system sorted according to the angle they form with the hand main direction.

Given the two sets $\mathcal{X}_L$ and $\mathcal{X}_D$, the best roto-translation $(R^*, \mathbf{t}^*)$ is the one solving the registration problem of Eq. 5.35:

$$(R^*, \mathbf{t}^*) = \underset{(R,\mathbf{t})}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j=1}^{5} ||R\mathbf{X}^i_{L,j} + \mathbf{t} - \mathbf{X}^i_{D,j}||^2 \tag{5.35}$$

namely, Eq. 5.35 consists in finding the best roto-translation that brings the point cloud $\mathcal{X}_L$ to the point cloud $\mathcal{X}_D$, and can be solved by the Horn's algorithm [62] enclosed in a RANSAC framework.

The test of Chapter 7 prove that the assumption of considering as fingertips $X^i_{D,j}$ the extreme point of the fingers is rather valid and that the mean error obtained from the square root of Eq. 5.35 is around $9mm$.

After the calibration has been performed, the hand centroid coordinates estimated by the Leap Motion are transformed into the range camera coordinate system obtaining the point $\mathbf{C_D} = \mathbf{RC_L} + \mathbf{t}$ and used as a starting point for the hand detection with the algorithm described in Chapter 3. Note how, although $\mathbf{C_D}$ is located in the hand region, its localization is not too accurate due to the uncertainty in the position estimated from the Leap Motion. For this reason $\mathbf{C_D}$ should not be directly used as the palm centroid, but its position must be optimized again with the circle or ellipse fitting scheme of Chapter 4.

The hand orientation vectors $\mathbf{h}$ and $\mathbf{n}$ provided by the Leap Motion can be directly used as hand orientation vectors in the depth camera system after a simple rotation ($\mathbf{x} = R\mathbf{h}$ and $\mathbf{z} = R\mathbf{n}$). It is worth noting that, although the hand orientation was ratherly well estimated by PCA, the hand direction was supposed to always point upward, while with the proposed approach this assumption can be removed relying in the direction estimated by the Leap Motion. Moreover, the hand orientation computed by the Leap Motion software proved to be more accurate than the one estimated with PCA.

5.4 Color features

Color features describe important textural characteristics of the segmented hand and, when the low-cost range cameras did not enter the mass market yet, were often employed by earlier automatic gesture recognition approaches in literature.

The color feature extraction algorithms implemented in the proposed framework are based on the scheme of Fig. 5.22 or its slight variations.

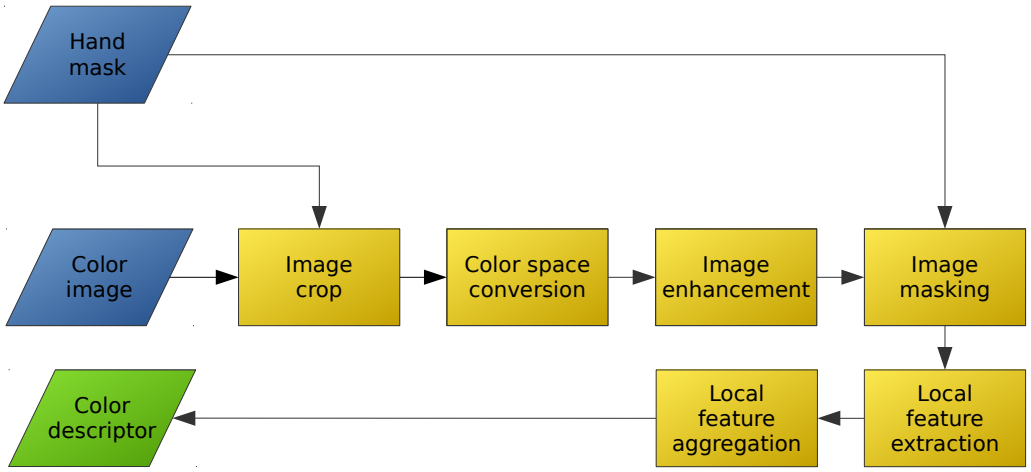


Figure 5.22: Generic pipeline of the employed feature extraction algorithm

Image cropping exploits the hand binary mask B_A, obtained from the arm removal of Section 4.3, to extract an image patch from the acquired color image limited to the region in B_A defined by the smallest bounding-box enclosing the hand pixels.

Color space conversion is another important preliminary step needed to change the common RGB(A) pixel representation of the color image, returned by most of the low-cost color cameras, to a more suitable one for the description of the textural properties of the cropped image. Recall from Section 3.2 that, in fact, while the RGB color space offers a more human-friendly representation of the pixel color, its performance in several computer vision tasks are rather low. Often the selected color space is Lab, as for the hand detection of Section 3.2, although the color descriptors extracted in the following sections make only use of the L channel as they were originally designed for grayscale images.

Then, often the gamma and contrast are normalized or adjusted in order to expand the low dynamic or compress the high one of the pixel intensity variations in the cropped image. This step is both needed to highlight primitive structures like edges and to minimize the dependency of the feature extraction algorithm

from the intensity levels. Image histogram equalization and image sharpening are image enhancement techniques commonly used in this step.

Image masking, exploiting again the depth mask B_A, prevents the color descriptor from being biased by the information coming from the background pixels contained in the cropped image. Since most feature extraction techniques aggregate the descriptors computed for each pixel, e.g. by creating a histogram of the descriptors distribution in the whole image or within limited regions, the masking avoids to account for the contributions of the retained background pixels.

The final steps in the pipeline of Fig. 5.22 consist in computing a color descriptor for each pixel, which is usually a local descriptor encoding the textural information within a texture patch centered on the pixel, and in aggregating all the pixel descriptors in a unique feature vector characterizing the underlining image. As already stated, the pixel descriptors are often collected in normalized histograms representing the descriptor distribution within the processed image.

5.4.1 Histogram of oriented gradients (HOG)

The histogram of oriented gradients (HoG) [63] is a textural descriptor for images widely used for people or object detection purposes (e.g., pedestrians or vehicles) on the acquired images or videos. It is based on the idea of dividing the image into small connected regions, called *cells*, and building for each cell a histogram of gradient directions for the considered pixels. The combination of these histograms then represents the descriptor. For a better invariance to changes in illumination or shadowing, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called *block*, and then using this value to normalize all the cells it contains.

The rationale is that the object shapes are well characterized by the distribution of local intensity gradients, namely by the occurrences of the gradient orientations in localized portions of an image. The implemented HoG feature extraction algorithm [64] follows the pipeline of Fig. 5.23, inspired by [63].

The first step, namely the color image normalization of its colors and gamma, is indeed optional thanks to the histogram normalization in the final steps.

Next, the grayscale conversion of the color image is only required for the intensity gradient computation, and in the current implementation consists in converting the acquired image in RGB(A) color space to CIELab and extracting the L component, associated to the lightness of the colors.

The image crop then, performed on the color image according to the hand mask B_A computed in the previous chapters, limits the descriptor computation

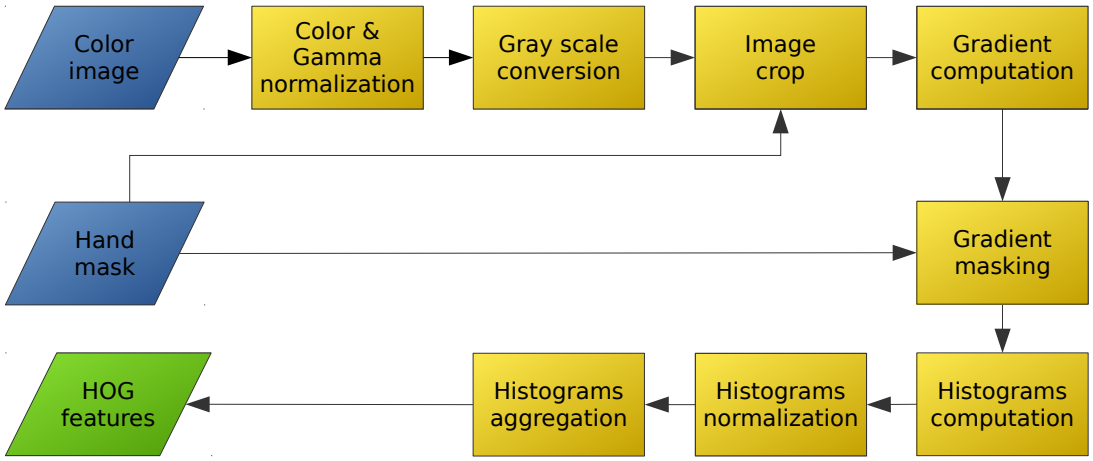


Figure 5.23: Adopted histogram of oriented gradients feature extraction pipeline

only within the minimum hand bounding box enclosing the hand pixels in B_A.

Although the further gradient computation on the cropped grayscale image can be performed accurately with Sobel or other discrete differentiation operators, the horizontal (G_x) and vertical (G_y) image gradients in the pipeline of Fig. 5.23 are computed by convolving the grayscale image patch I_G with separable 1-D kernels $g^x_{u,v} = [-1 \quad 0 \quad 1]$ and $g^y_{u,v} = [-1 \quad 0 \quad 1]^T$ [63]. The gradient magnitude for each pixel $m_{u,v}$ is then computed as $m_{u,v} = \sqrt{{g^x_{u,v}}^2 + {g^y_{u,v}}^2}$. Differently from the original algorithm of [63], the implemented version adds an intermediate step which consists in forcing the gradient magnitude to 0 for the cropped image pixels discarded by the binary mask B_A. This expedient prevents the descriptor from being biased by the intensity of the background pixels.

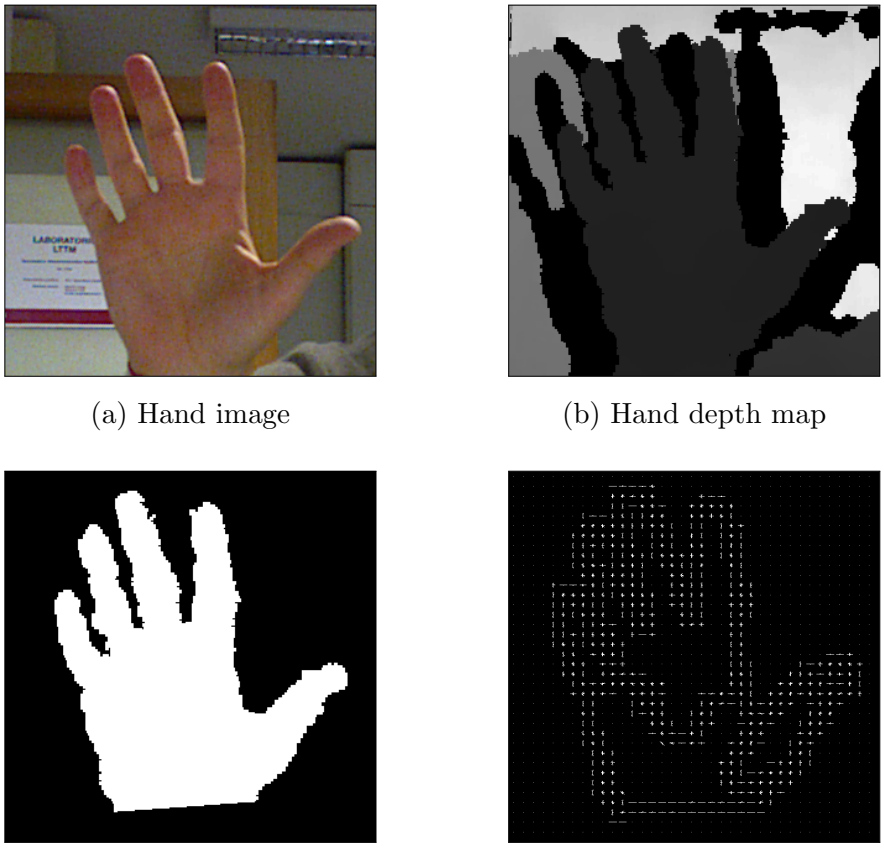
The grayscale image is then partitioned into a grid of $M \times N$ rectangular *cells* of size $C_H \times C_W$ pixels. Note how the cell size depends on the cropped grayscale image width and height.

Each cell will generate a separate histogram of B bins, each one centered on a given gradient direction from 0 to 180° or 0 to 360° depending on the usage of the unsigned or signed gradients. Each pixel within the cell $C_{i,j}$ will cast a vote either corresponding to its gradient magnitude or to the value of a proper function of the gradient magnitude (e.g., square root, square or a clipped version of the magnitude). The current implementation exploits the gradient magnitude itself, as it outperforms the other functions [63]. Since the gradient directions are real numbers, they do not necessarily correspond to an histogram bin center, thus a unique assignation to a bin is not possible. A possible solution consists in using *linear interpolation* to split each pixel vote to the interested bins, where

the fraction of the assigned vote is in an inverse proportion with the distance of the (real) gradient direction value from the neighboring bin centers.

For a better invariance to illumination and contrast, the cells are grouped together in larger and spatially connected *blocks*, and the histograms of the cells within the same block are locally normalized. Blocks of cells may either adopt a rectangular (R-HOG) or radial (C-HOG) geometry, and may be either separated or overlapped. In the latter case, each block shares a certain number of cells with the surrounding blocks. The experiments in Chapter 7 use separate rectangular blocks of 5×6 cells, where each cell generates a histogram with $B = 9$ bins accounting for gradient directions from 0 to 180° with a width of 20° [64].

The final HoG descriptor consists in the concatenation of the normalized histograms within each block, and the related feature vector is denoted by $\mathbf{F}^{\mathbf{hog}}$. It is worth noting that, in case of overlapping blocks, the feature vector will contain repetitions of the same histograms with different normalization. Fig. 5.24 shows an example of HoG descriptor extraction from a given hand image.



(a) Hand image (b) Hand depth map

(c) Hand depth mask (d) Hand HoG features

Figure 5.24: Example of HoG descriptor extraction for a given hand image

5.4.2 Local phase quantization (LPQ)

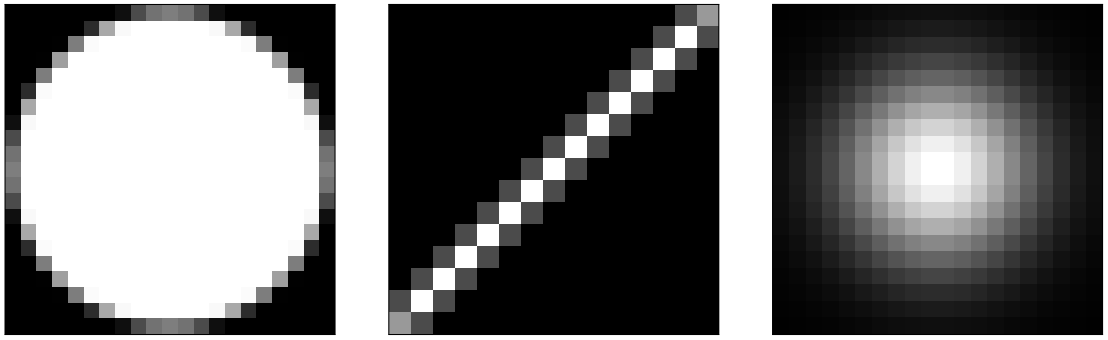
Textural information gives a noteworthy contribution in object and subject recognition, like the automatic hand gesture recognition in this thesis, and was the only available cue in several pattern recognition methods in the earlier literature.

Real color cameras, as already stated in Section 2.1, are affected by several image degradation problems due to the low quality camera sensor or optics, lens distortion or misalignment, object and subject motion respect to the camera and unfavorable lighting conditions. Image degradation, beside making the acquired images visually unpleasant, often leads several computer vision algorithms to fail.

One of he most commonly encountered degradations is *blurring*, which may arise from:

- out of focus framed scene
- motion of the objects or subjects respect to the camera
- atmospheric turbulence

Image blur for a grayscale image $I_G = \{i^G_{u,v}\}$ can be ideally described by the convolution of I_G with a *point spread function* (PSF) or kernel $K = \{k_{u,v}\}$ modeling the blur type. Fig. 5.25 compares three kernels modeling an out of focus (Fig. 5.25(a)), motion (Fig. 5.25(b)) and atmospheric turbulence (Fig. 5.25(c)) blurs, corresponding to an Airy disk, a line and gaussian kernel respectively.



(a) Out of focus blur (b) Motion blur (c) Turbulence blur

Figure 5.25: Comparison of common image blur models

It is worth noting that the image blur modeling defined above is only valid for ideal blurs, that are *spatially invariant*, namely when the blur model (kernel) does not depend on the image region the kernel is convolved with. The ideal blur invariance is, however, not achieved in practice.

Local Phase Quantization [65] is an image descriptor based on the fact that texture is a *local* image property, and is designed to be robust against the most common image blurs described in the previous lines. LPQ assumes that the point spread function of the blur can be *locally* approximated by a centrally symmetric model, which is often sufficient for example in the case of camera motion, misfocused optics, and atmospheric turbulence.

Assuming the selected PSF is centrally symmetric, from the properties of the Fourier transform it follows that the *phase* of the filtered image corresponds to the phase of I_G in the frequency domain for *positive* values of the magnitude of the Fourier transform of the PSF [65]. This property always holds for the transforms of certain blur models (e.g. the transform of the Gaussian kernel used to model the atmospheric turbulence blur is always positive by definition), while for other blur types (e.g. out of focus and ideal motion blurs) modeled by sinc or Bessel functions the property holds up to the first zero-crossing frequency. Moreover, since most of the blur models show a low-pass filtering behavior and their energy is concentrated in the Fourier coefficients corresponding to the lowest frequencies, the PSF frequency response support can be limited to the first zero-crossing frequency without a significant information loss. This expedient allows, if needed, to approximate the PSF frequency response with a positive values function to maintain the blur invariance property of the Fourier phase spectrum.

Based on the previous rationale, LPQ exploits the local phase information contained in the phase spectrum of a texture patch of size $m \times m$ around a pixel in position (u, v) in I_G for characterizing the underlining image texture, since for the blur invariance the possible image blur does not corrupt the texture phase. The implemented version of the LPQ descriptor extraction approach, schematized in Fig. 5.26, is based on the original work of [65].

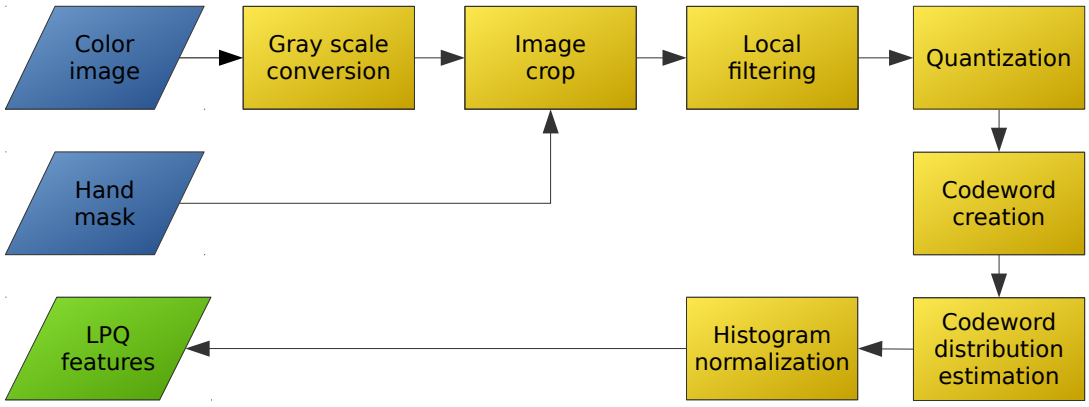


Figure 5.26: Adopted local phase quantization feature extraction pipeline

The first steps in the pipeline of Fig. 5.26 are in common with the pipeline of the texture descriptor of Section 5.4.1 and simply consist in converting the acquired color image in grayscale and extract the hand region with the aid of the hand mask obtained in the previous chapters. Note how the hand mask may, again, be also used to avoid the computation of the LPQ features for the background pixels within the extracted hand patch.

Then, the method computes for each pixel $i^G_{u,v}$ the Fourier transform within a texture patch of size $m \times m$ with center in (u, v). While the Discrete Fourier Transform (DFT) is suitable when applied to the whole image, in this case the Short-Time Fourier Transform (STFT), widely used in signal processing, is more appropriate since the spectrum is only computed within a limited patch.

Let $F(i^G_{u,v}, \mathbf{f})$, with $\mathbf{f} = [f_u \quad f_v]$ denoting the horizontal and vertical frequency vector, be the STFT of the texture patch centered on the image pixel at position (u, v). Note how $\mathbf{f} \in \{\mathbf{f_1}, \mathbf{f_2}, \ldots, \mathbf{f_L}\}$ with L the support length of $F(i^G_{u,v}, \mathbf{f})$ which corresponds to the support length of the frequency response of the positive values of the PSF in order to maintain the blur phase invariance property. While the concatenation of all the $F(i^G_{u,v}, \mathbf{f})$ phase vectors could be directly used as a texture descriptor, its length and the space required to encode the real phase values would make it impractically usable.

For this reason, the LPQ algorithm first performs a quantization of the SFTF phase according to Eq. 5.36.

$$Q(F(i^G_{u,v}, \mathbf{f})) = \text{sgn}(\Re F(i^G_{u,v}, \mathbf{f})) + 2\,\text{sgn}(\Im F(i^G_{u,v}, \mathbf{f})) \tag{5.36}$$

where $Q \in \{0, 1, 2, 3\}$, $\Re(\cdot)$ and $\Im(\cdot)$ denote respectively the real and imaginary part of the frequency response and, the sign function this time assigns 1 to positive values and 0 to the negative ones. It is worth noting that Eq. 5.36 does not require an explicit computation of the phase angle. The L coefficients have been previously decorrelated [65] since correlated coefficients would not carry relevant information.

The quantized phase values for the L frequencies are then represented by 2-bit codewords aggregated in a unique codeword $C_W(i^G_{u,v})$ of $2L$ bits (Eq. 5.37) which ranges from 0 to $2^{2L} - 1$.

$$C_W(i^G_{u,v}) = \sum_{i=1}^{L} Q(F(i^G_{u,v}, \mathbf{f_i}))2^{i-1} \tag{5.37}$$

After computing the codewords encoding all the local patches, the method builds an histogram with $2^{2L} - 1$ bins (one per codeword) representing the distribution of the codewords for the image I_G. The histogram is then normalized in order to make the sum of each scaled codeword frequency value unitary [65].

The extracted descriptor corresponds, then, to the normalized histogram of the codeword frequencies. In the experiments of Chapter 7, instead, the final descriptor is made by the concatenation of two codeword distribution histograms computed for texture patches of size 3×3 and 5×5 in order to carry an higher amount of information on the pixel neighborhoods [66, 64], analogously to the curvature distribution histograms in Section 5.1.4. The rationale is that when using small values of m the lower frequencies capture more details of the underlining image patch but reduce the insensitivity of the method to the image blur, while higher values of m lead to a better blur insensitivity but reduce the descriptor discrimination capability.

5.4.3 Local ternary patterns (LTP)

Local Ternary Patterns (LTP) [67] is another computationally efficient nonparametric local image descriptor widely used in face detection algorithms [68].

It is based on encoding the information contained in a texture patch of size $m \times m$ with a codeword that will be either used as a single feature or aggregated in local histograms representing the codeword distribution in a wider region. This descriptor is a generalization of the Local Binary Pattern (LBP) [69] with an higher discrimination capability and a lower sensitivity to noise in uniform regions.

Both LBP and LTP extraction algorithms consist in a binary thresholding of the pixel intensities in the neighborhood (usually a 8-neighbors) on the basis of the intensity of the central pixel of the texture patch, followed by the encoding in a binary or ternary codeword composed by concatenating the binary or ternary digits associated to the neighboring pixels.

The LBP and LTP codeword extraction is formalized in Eq. 5.38.

$$\begin{aligned} LBP(u_c, v_c) &= \sum_{n=0}^{7} 2^n s_b(i_n, i_c) \qquad \text{with } s_b(i_n, i_c) = \begin{cases} 1 & \text{if } i_n \geq i_c \\ 0 & \text{otherwise} \end{cases} \\ LTP(u_c, v_c, t) &= \sum_{n=0}^{7} 3^n s_t(i_n, i_c, t) \quad \text{with } s_t(i_n, i_c, t) = \begin{cases} 1 & \text{if } i_n \geq i_c + t \\ 0 & \text{if } |i_n - i_c| < t \\ -1 & \text{if } i_n \leq i_c - t \end{cases} \end{aligned} \tag{5.38}$$

where i_c and i_n denote respectively the intensities of the central pixel of the image patch and the one of the n-th pixel in its neighborhood, $s_b(\cdot)$ and $s_t(\cdot)$ respectively a binary and a ternary thresholding functions.

It is worth noting that, because of $s_b(\cdot)$, LBP is highly sensitive to the image noise since very high or very low pixel intensities due to noise may corrupt the extracted codeword. LTP, instead, thanks to the global threshold t that accounts for the image noise, is less sensitive to the codeword corruption due to local fluctuations of the pixel intensities caused by noise.

Moreover, both in this work and the original version of [67] the LTP descriptor computation is actually decomposed in the extraction of two LBP descriptors, where the second one is obtained by inverting the thresholding function $s_b(\cdot)$ (Fig. 5.27). The two sub-descriptors are then only aggregated in the final phase.

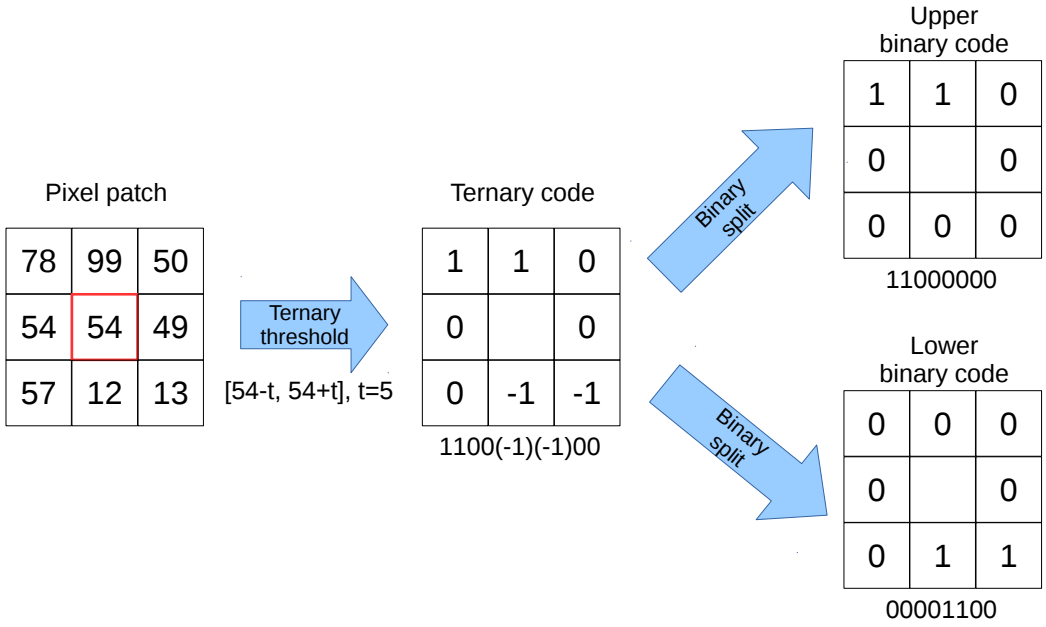


Figure 5.27: LTP descriptor split in two LBP codewords

Fig. 5.28 reports the pipeline of the used LTP feature extraction algorithm.

The first steps in the pipeline of Fig. 5.28 consist in the extraction of the hand region from the grayscale image using the information provided by the hand depth mask, followed by a few pre-processing steps aimed at enhancing the local dynamic range of the image in dark or shaded regions while compressing it in bright areas (gamma correction) and augmenting the edges contrast with techniques of image sharpening based on the difference of gaussians (DoG).

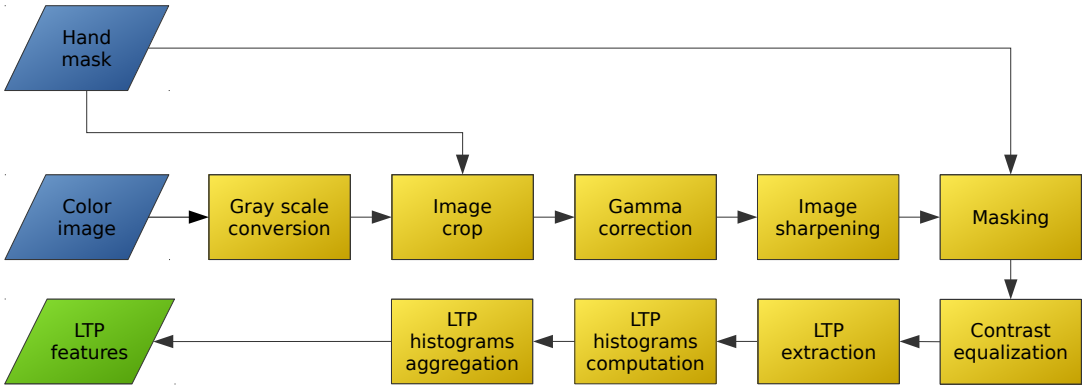


Figure 5.28: Pipeline of the LTP descriptor extraction algorithm

Then, the enhanced image is masked with the previously defined hand binary depth mask in order to avoid the extraction of codewords of pixels referred to the background that may disrupt the final descriptor.

The final pre-processing step consists in equalizing the image contrast in order to make the extracted descriptor independent from the high or low pixel intensity dynamic range variations among different frames.

A pair of dynamic codewords is then computed for each non-masked pixel in the hand region with the approach exemplified in Fig. 5.27 and a codeword histogram is generated for each texture patch of the image partitioned in a grid. The final descriptor consists in the concatenation of the normalized histograms of each image patch. Note how, indeed, the descriptor is made by the juxtaposition of the feature vectors generated by the two LBP sub-descriptors.

The current work evaluates two variations [64] of the original approach of [67]: the first variation defines neighborhoods of different size (eg., 8-neighborhood and 16-neighborhood in the tests of Chapter 7) to deal with textures at varying scales, while the second one only adopts *uniform* binary codewords exactly containing at most one transition 0-1 and one 1-0 which represent primitive structural information like edges and corners.

Chapter 6

Feature classification

The third and last main step in the gesture recognition pipeline of Section 1.3 consists in classifying the feature vectors extracted in Chapter 5 in order to reliably recognize the performed gestures. Classification makes use of a gesture recognition model previously computed on a feature vector set, extracted from a given dataset containing several repetitions (e.g. matched color images and depth maps) performed by different people of gestures from the selected dictionary, with a proper machine learning technique.

Approaches based on Support Vector Machines, Decisional Trees, Neural Networks, genetic algorithms and many others have been proposed in literature to tackle automatic gesture recognition and several other computer vision problems. Presenting a complete taxonomy of the various machine learning methods in literature, though, is beyond the scopes of this chapter, that limits the analysis to the classification algorithms used for the tests of Chapter 7. A more thorough treatment of the classification problem can be found in [70].

The chapter is articulated as follows: Section 6.1 and 6.2 shortly describe two single classifiers that have been employed for several years for the solution of computer vision tasks due to their computational efficiency and broad diffusion. Section 6.3 shows how more complex and robust classifiers can be built by assembling several learners, both of the same or different types. Section 6.4 reports a few useful algorithms that may be employed to extract a minimal subset of most relevant features which boost the classification performance without sensibly affecting the estimated classification model accuracy. Eventually, the chapter presents a few metrics used to compare the actual performance of the considered classifiers in this work.

6.1 Support vector machines (SVM)

Support Vector Machines [23] are linear binary classifiers used both for classification and regression tasks. Given a training set $\mathcal{X}$ of N vectors $\mathbf{x_i} \in \mathbb{R}^F, i = 1, \ldots, N$ of F features and a vector of labels $\mathbf{y} \in \{1, -1\}^F$, SVM finds the best hyperplane separating the feature vectors in the feature space $\mathbb{R}^F$ by solving the quadratic optimization problem of Eq. 6.1.

$$\begin{aligned} (\mathbf{w}^*, b^*, \xi^*) &= \underset{(\mathbf{w},b,\xi)}{\operatorname{argmin}} \left\{ \frac{1}{2}\mathbf{w}\mathbf{w}^T + C\sum_{i=1}^{N} \xi_i \right\} \\ \text{subject to } & y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, i = 1, \ldots, N \end{aligned} \tag{6.1}$$

where C is a penalty parameter on the training error. For any testing vector $\mathbf{x}$, after the estimation of the best $(\mathbf{w}, b, \xi)$ the predictor assigns a class to $\mathbf{x}$ according to Eq. 6.2.

$$f(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}^T\mathbf{x} + b) \in \{-1, 1\} \tag{6.2}$$

Note how the result of Eq. 6.2 is a value indicating in which of the two semi spaces of $\mathbb{R}^F$ separated by the hyperplane defined by $\mathbf{w}$ and b the feature vector $\mathbf{x}$ is located. Usually the best hyperplane is the one maximizing its distance from each separated sample. An example of separating hyperplanes defined by $\mathbf{w}$ and b in a bidimensional space is shown in Fig. 6.1.

The original SVM implementation, though, may be only used in case of linearly separable samples. Several computer vision tasks, including automatic gesture recognition present, instead, feature vectors living in highly nonlinear feature spaces, that are then not separable by one or more hyperplanes. For this reason, SVM has been extended with the "kernel trick" in order to map the feature vectors $\mathbf{x_i}$ of a non linear space to an higher dimensional space where their transformed versions are linearly separable. Eq. 6.1 and 6.2 are, then, slightly modified in Eq. 6.3 and 6.4 respectively to take into account the feature mapping.

$$\begin{aligned} (\mathbf{w}^*, b^*, \xi^*) &= \underset{(\mathbf{w},b,\xi)}{\operatorname{argmin}} \left\{ \frac{1}{2}\mathbf{w}\mathbf{w}^T + C\sum_{i=1}^{N} \xi_i \right\} \\ \text{subject to } & y_i(\mathbf{w} \cdot \phi(\mathbf{x_i}) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, i = 1, \ldots, N \end{aligned} \tag{6.3}$$

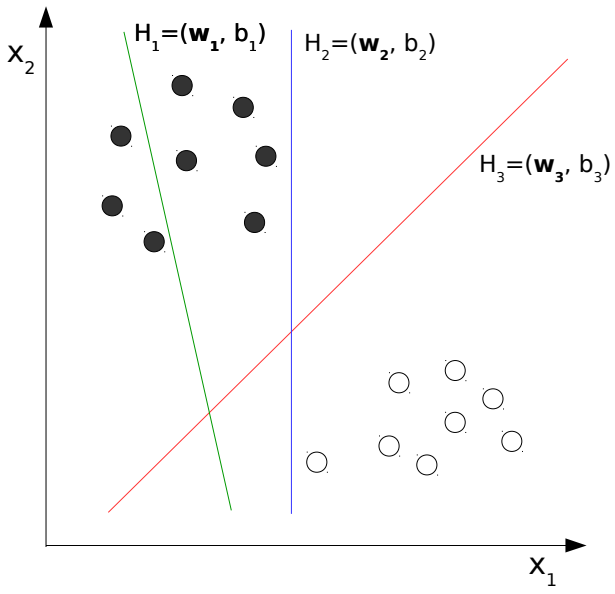


Figure 6.1: Example of SVM separating hyper planes: non separating (H_1), separating with a low margin (H_2), separating with a high margin (H_3)

$$f(\mathbf{x}) = sgn(\mathbf{w} \cdot \phi(\mathbf{x}) + b) \in \{-1, 1\} \tag{6.4}$$

where $\phi(\mathbf{x})$ denotes the map from the original feature space to the higher dimensional transformed space. This rationale is depicted in Fig. 6.2.

Note how the transform $\phi(\mathbf{x_i})$ is related to a selected kernel function $k(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j})$ and the dot product $\mathbf{w} \cdot \phi(\mathbf{x})$ is again expressed in terms of the kernel function $\mathbf{w} \cdot \phi(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i k(\mathbf{x_i}, \mathbf{x})$, with $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x_i})$ in the transformed space. The map $\phi(\cdot)$ does not change the original SVM algorithm but only modifies the input vectors $\mathbf{x_i}$. Operationally, the dot product $\mathbf{w} \cdot \phi(\mathbf{x})$ can be visualized as $\mathbf{w}^T \mathbf{x}'$ with $\mathbf{x}'$ the projected vector $\mathbf{x}$ in the linear feature space.

Several kernel functions are present in literature:

- **Linear kernel:** $k(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^T \mathbf{x_j}$
- **Polynomial kernel:** $k(\mathbf{x_i}, \mathbf{x_j}) = (\gamma \mathbf{x_i}^T \mathbf{x_j} + \delta)^n$ with n degree of the polynomial
- **Radial kernel:** $k(\mathbf{x_i}, \mathbf{x_j}) = C \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|)^2$
- **Sigmoid kernel:** $k(\mathbf{x_i}, \mathbf{x_j}) = C \tanh(\gamma \mathbf{x_i}^T \mathbf{x_j} + \delta)$

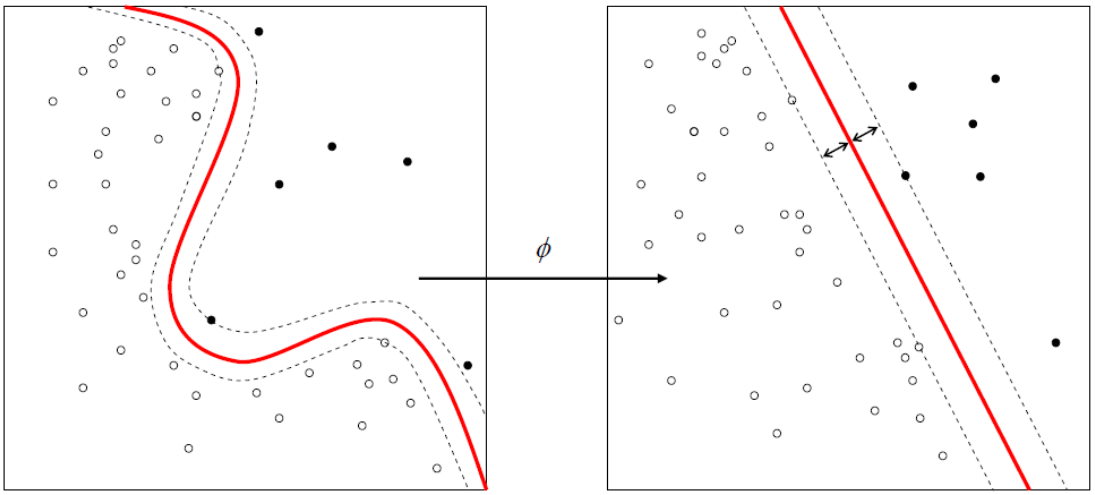


Figure 6.2: Feature vector mapping in a kernel SVM classifier

The radial kernel, though, is the most used in several classification methods based on SVM.

As stated previously, Support Vector Machines were originally binary classifiers and are therefore not directly usable as predictors in multi-class classification tasks like the automatic gesture recognition presented in this thesis. More recent implementations extended the original method of [23] to the multi-class classification and regression case. Some of them consist in decomposing the multi-class classification problem in multiple independent binary classification tasks, e.g., the "one against one" approach of [71] trains a set of $K(K-1)/2$ binary SVM classifiers used to test all the K classes against each other. Each classification output is then chosen as a *vote* for a certain class and the class that obtained the maximum number of votes is assigned to the input feature vector. This is the method implemented by LibSVM [72], widely employed in the proposed gesture recognition framework (with $K = G$ cardinality of the gesture dictionary in this case).

Both for the binary and the multi-class extension of SVM, setting the proper values for the selected kernel parameters is not a trivial task, as slight parameter variations may sometimes lead to high differences in the estimated model accuracy on the same dataset. Grid Search [72] is de facto standard way of performing the kernel parameter optimization, and consists in a simple exhaustive searching through a manually specified subset of the parameter space. The algorithm is driven by a proper performance metric for the tested parameter values, which is usually the *cross-validation* on the training set. A naive implementation of Grid Search simply tests every possible parameter combination value, selecting

the one maximizing the performance metric. As each combination can be tested independently from the others, the best parameter search is highly parallelizable but continues to suffer from the "curse of dimensionality". The work in this thesis uses a radial kernel for the gesture recognition purposes, as it offered the highest performance among all the previous kernels. In this case, the search space for the Grid Search algorithm is bidimensional, since the radial kernel only requires the optimization of two parameters.

Another serious problem in parameter optimization, especially with training sets of low cardinality, is the high risk of *overfitting*, namely the low generalization on the trained model. The model trained with the estimated optimal parameters for a given training set, that is the parameter values maximizing the expected model accuracy may, in fact, have poor performance with new feature vectors not accounted in the parameter optimization. This is more likely to happen with small training sets, as they usually not contain exhaustive data to represent the variability of real data. Moreover, as most classification approaches require separate training and test sets, the problem is more serious in this case as the original dataset splitting further reduces the cardinality of the training set. While in certain situations the problem can be tackled by using or building new datasets with an higher cardinality, in other cases this is not possible, for example when the same small datasets are shared among various research groups to compare the performance of the respective gesture recognition algorithms. This is the case of the datasets employed in the experiments of Chapter 7.

For these reasons, this thesis proposes a new gesture recognition model training approach based on Grid Search and exploiting the whole dataset while reducing the over-fitting of the estimated model parameter values [54]. Consider a training set $\mathcal{D}$ containing data from N users. The main idea of the proposed learning protocol consists in training N independent sub-models M_i for $i = 1, \ldots, N$ and considering the average of their estimated accuracies as the overall gesture recognition approach accuracy. Let $\mathcal{P}_i \subset \mathcal{D}$ denote the subset of the feature vectors of dataset $\mathcal{D}$ extracted from the person's P_i data only. Each model M_i, referred to a separate person P_i in the dataset, is trained on the subset $\mathcal{T}_i = \mathcal{D} \setminus \mathcal{P}_i$ and validated on $\mathcal{V}_i = \mathcal{P}_i$, namely the feature vectors associated to person P_i are only used to test the accuracy of the model trained on the other people data in the dataset. This rationale ensures the training of each separate model M_i is not biased by the selected validation data, and offers a better generalization as the whole dataset is exploited by varying the training and validation sets.

The proposed training protocol, exemplified in Fig. 6.3, estimates the best

parameter values for the radial kernel of a SVM classifier trained on $\mathcal{T}_i$ of each submodel M_i. Such estimation exploits an ad-hoc variation of the classic Grid Search approach named "leave-one-person-out": while the classic implementation of Grid Search selects the best parameter values maximizing the cross-validation on the set $\mathcal{T}_i$, which is based on K random partitioning (a common value for K is 5) of $\mathcal{T}_i$ in a training and a validation sets, the proposed variation uses $N-1$ pre-defined partitioning following the same rationale of the submodel M_i definition. More specifically the method, for each training set $\mathcal{T}_i$ and each parameter pair (γ, C), trains $N-1$ submodels $M_{i,j}, j = 1, \ldots, N-1$ on the subsets $\mathcal{T}_{i,j} = \mathcal{T}_i \setminus \mathcal{V}_j$ with $\mathcal{V}_j = \mathcal{P}_j \subset \mathcal{T}_i$ and selects the best pair (γ_i^*, C_i^*) maximizing the average validation accuracy on the subsets $\mathcal{V}_{i,j}$.

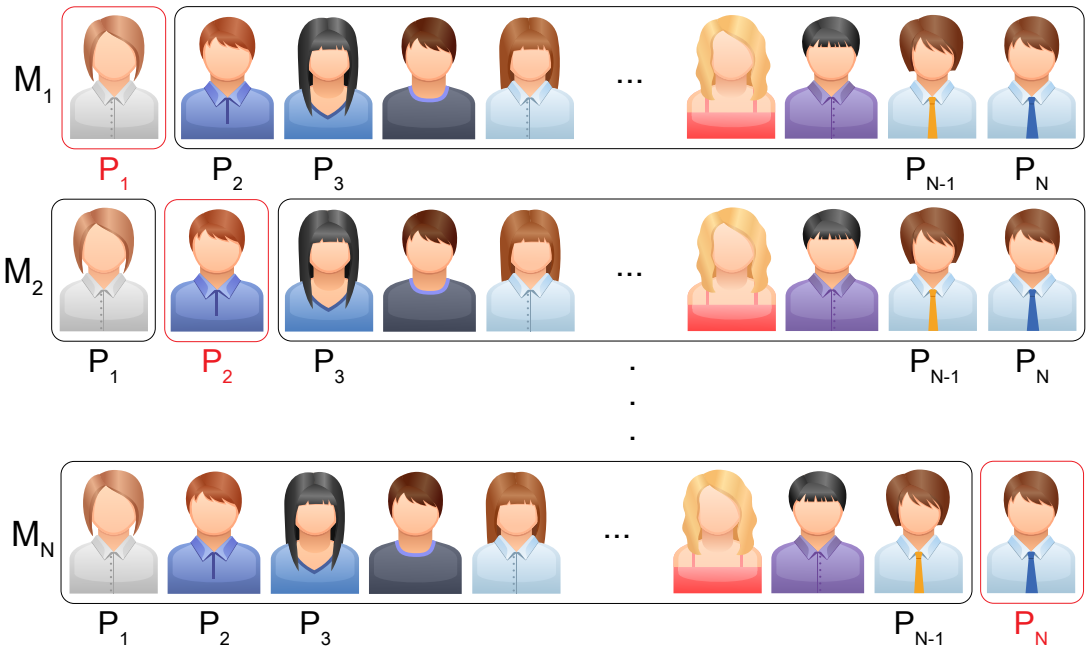


Figure 6.3: Leave-one-person-out approach

It is worth noting that the approach of Fig. 6.3 may be only used to estimate the theoretical accuracy of the gesture recognition approach from a small dataset describing the gesture dictionary of interest. The final model M, to be used in a given gesture recognition application, has to be trained using the same approach used to optimize the parameters for each model M_i replacing $\mathcal{T}_i$ with $\mathcal{D}$.

6.2 Random forests

Random Forests [26] is a powerful classifier based on growing several classification trees each one voting for a certain class. Given a new feature vector to classify, each tree in the forest votes for a class and the class obtaining the majority of the votes will be assigned to to the input vector.

Each tree is grown as follows:

- If the training set $\mathcal{T}$ contains N feature vectors $\mathbf{x_i} \in \mathbb{R}^F$, sample N vectors randomly with replacement from $\mathcal{T}$ to build the training set $\mathcal{T}_b$ for the b-th tree in the forest. Note how $\mathcal{T}_b$ is more likely to contain several repeated vectors.

- Select a number $f << F$ of features such that at each node in a given tree f features are sampled randomly from the feature space, and the *best split* from them is used to split the node. Note how f is held constant during the forest growing.

- Each tree is grown to the largest extent possible. No pruning criteria is used for early terminating its growth.

Random Forests is, then, a variation of the *bootstrap aggregating* [73] (or "bagging") technique for tree learners. While tree bagging determines the best split at each node of the tree from the whole feature set, Random Forests only use a random subset of features (feature bagging). This expedient reduces the effect of a serious draw-back of tree bagging, that is the *tree correlation*: features acting as strong predictors in bagged trees are more likely to be select for node splitting in each tree, that are then highly correlated and do not carry any useful information.

Analogously to tree bagging, increasing arbitrarily the number of trees in a Random Forest decreases the model variance (that is the sensitivity to fluctuations in the data) without affecting the bias, that is, the training and test errors tend to level off after training a sufficient number of trees, as shown in Fig. 6.4.

The Random Forests error rate depends, then, on two aspects:

- The correlation between any two trees in the forest. Increasing the correlation increases the forest classification error rate.

- The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

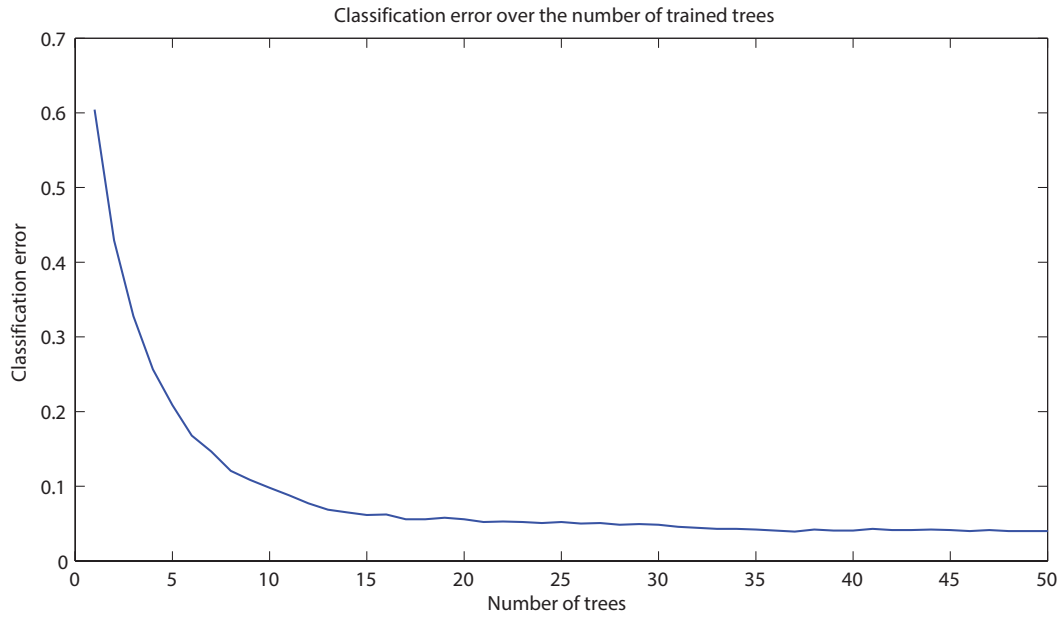


Figure 6.4: Example of out-of-bag error variation over the number of trained trees

Reducing the number of randomly sampled features f at each node reduces both the correlation and the strength, while increasing it increases them both. The optimal value of f is a trade-off between the tree mutual correlation and the classifier strength trained on each tree. The evaluation of the out-of-bag error rate is a good way to detect a suitable value for f. Note how f is the only adjustable parameter to which random forests is somewhat sensitive, compared to the two or more parameters for an SVM classifier.

When the training set for a given tree in the forest is drawn by sampling with replacement, about one-third of the feature vectors are left out of the sample. This oob (out-of-bag) data is used to get a running unbiased estimate of the classification error as trees are added to the forest. Namely, the oob data constitutes the validation set for the considered tree in the forest. More specifically, after training the trees on their separate training sets $\mathcal{T}_b$, for each feature vector $\mathbf{x_i}$ of the full training set $\mathcal{T}$ $\mathbf{x_i}$ is assigned the class which received the highest number of votes accounted only from the trees for which $\mathbf{x_i}$ was an out-of-bag feature vector (namely, the trees for which $\mathbf{x_i}$ was not selected as a training vector). The out-of-bag classification error is, then, the ratio of the misclassified feature vectors $\mathbf{x_i}$ over the full training set size N, where each $\mathbf{x_i}$ was only tested by the subset of trees in the Random Forest for which $\mathbf{x_i}$ was an oob feature vector.

Note how the out-of-bag error is also used to get estimates of a feature importance, as will be described more in detail in Section 6.4.3.

6.3 Ensembles of classifiers

Several machine learning approaches, especially the earliest proposed in literature, tackle the classification problem by training a *single* classifier on given dataset $\mathcal{D}$. The classifier parameters are usually previously tuned in order to maximize a performance metric, e.g. the estimated classification accuracy of the model built on $\mathcal{D}$. Machine learning based on single classifiers is, however, affected by several problems, for example:

- **Feature vector length:** the computational load required to train a classification model increases dramatically with the length of the feature vectors in $\mathcal{D}$.
- **Classification bias:** the classification model trained on $\mathcal{D}$ may suffer of an high over-fitting, with a consequent sensible drop on the classification accuracy of new data (feature vectors not contained in $\mathcal{D}$).

The first problem can be tackled by training a set of independent *homogeneous* classifiers [74], each one considering only a subset of the features in $\mathcal{D}$, and formulate the final decision by combining their individual opinions to derive a consensus response. The rationale is reducing the computational complexity by partitioning the original problem in several independent sub-problems, which may be solved in *parallel*.

The second problem can be solved with the same approach but exploiting an ensemble of *heterogeneous* classifiers, each one capable of capturing different relevant properties of the selected features. Other classification schemes adopt ensembles of classifiers trained on different datasets or even ensembles of ensembles of classifiers in a hierarchical fashion.

Consider a generic pattern recognition problem where a feature vector $\mathbf{x}$ has to be assigned to one of the K possible classes $c_1, \ldots, c_K$. Assume also to have R classifiers, each one trained on a different subset of features of $\mathbf{x}$, with $\mathbf{x_i}$ for $i = 1, \ldots, R$ feature vector for the i-th classifier. Note how $\mathbf{x_i}$ may denote, instead, a different measurement vector on the same pattern in case of heterogeneous classifiers. In the measurement space each class c_k can be modeled by its distribution $P(\mathbf{x_i}|c_k)$ among the feature vectors of the dataset and its *a priori probability* of occurrence denoted by $P(c_k)$. According to the Bayesian theory, given the measurement vectors $\mathbf{x_i}, i = 1, \ldots, R$, $\mathbf{x}$ should be assigned by the ensemble to the class c maximizing the *a posteriori probability* of that interpretation (Eq. 6.5).

$$c = \arg\max_{c_k} P(c_k|\mathbf{x_1}, \mathbf{x_1}, ..., \mathbf{x_R}) \tag{6.5}$$

Eq. 6.5 states that, in order to exploit all the available information to correctly make a proper decision, it is essential to compute the probabilities of the various hypotheses (namely, the assignment of $\mathbf{x}$ to another class) by considering all the measurements $\mathbf{x_1}, \ldots, \mathbf{x_R}$ simultaneously. This is theoretically correct, although impractically solvable as the computation of the a posteriori probability functions $P(c_k|\mathbf{x_1}, \ldots, \mathbf{x_R})$ would require the knowledge of the joint distributions $P(\mathbf{x_1}, \ldots, \mathbf{x_R}|c_k)$ which would be difficult to infer.

Practical applications of the rule of Eq. 6.5 attempt to simplify and express it in terms of intermediate decision support computations performed by the individual classifiers, each one exploiting only the information provided by the relative measurement vectors (or in this case, feature vectors) $\mathbf{x_i}$. The first step starts from the application of the Bayes theorem to the a posteriori probability $P(c_k|\mathbf{x_1}, \ldots, \mathbf{x_R})$ to express it in a more manageable form (Eq. 6.6).

$$P(c_k|\mathbf{x_1}, \ldots, \mathbf{x_R}) = \frac{P(\mathbf{x_1}, \ldots, \mathbf{x_R}|c_k)P(c_k)}{P(\mathbf{x_1}, \ldots, \mathbf{x_R})} \tag{6.6}$$

where $P(\mathbf{x_1}, \ldots, \mathbf{x_R})$ is the unconditional measurement joint distribution, which in turn can be expressed in terms of the conditional measurement distributions as in Eq. 6.7:

$$P(\mathbf{x_1}, \ldots, \mathbf{x_R}) = \sum_{k=1}^{K} P(\mathbf{x_1}, \ldots, \mathbf{x_R}|c_k)P(c_k) \tag{6.7}$$

Now, $P(\mathbf{x_1}, \ldots, \mathbf{x_R}|c_k)$ represents the joint probability distribution of the measurements extracted by the classifiers, and $P(c_k)$ is obtained from the dataset $\mathcal{D}$ statistics (e.g., $P(c_k)$ can be the ratio of the number of feature vectors belonging to class c_k in $\mathcal{D}$ and the dataset cardinality $|\mathcal{D}|$).

Assuming the classifiers in the ensemble are *conditionally statistically independent*, Eq. 6.6 may be rewritten as:

$$P(c_k|\mathbf{x_1}, \ldots, \mathbf{x_R}) = \frac{P(c_k)\prod_{i=1}^{R} P(\mathbf{x_i}|c_k)}{\sum_{j=1}^{K} P(c_j)\prod_{i=1}^{R} P(\mathbf{x_i}|c_j)} \tag{6.8}$$

By replacing Eq 6.8 in Eq. 6.5, the latter can be rewritten as Eq. 6.9 [75].

$$c = \underset{c_k}{\operatorname{argmax}} \left\{ P^{-(R-1)}(c_k) \prod_{i=1}^{R} P(c_k|\mathbf{x_i}) \right\} \tag{6.9}$$

where $P(c_k|\mathbf{x_i})$ is given by the i-th classifier (e.g., LibSVM[72] offers the computation of $P(c_k|\mathbf{x_i})$ as an option). Often the the a posteriori probabilities $P(c_k|\mathbf{x_i})$ computed by the respective classifiers do not deviate dramatically from the prior probabilities $P(c_k)$, especially in presence of noise in the measurements, rather common when using low-cost sensors. In this case, Eq. 6.9 may be rewritten as the *sum rule* [75] (Eq. 6.10)

$$c = \underset{c_k}{\operatorname{argmax}} \left\{ (1-R)P(c_k) + \sum_{i=1}^{R} P(\mathbf{c_k}|\mathbf{x_i}) \right\} \tag{6.10}$$

As showed in [75], there are several other decision rules for the class assignment alternative to the product rule (Eq. 6.9) and the sum rule (Eq. 6.10), although the sum rule may be proved theoretically that outperforms the other classifier combination schemes. For this reason, it is often use in in methods exploiting ensembles of classifiers (e.g., [74]).

The remaining part of this section describes briefly how different kinds of ensembles of classifiers have been used to improve the recognition accuracy of the gesture recognition approach in this work. Their performance are analyzed in Chapter 7.

6.3.1 Random subspace ensemble

Automatic hand gesture recognition is a difficult task not only for the hand detection and segmentation problems, but also for the classification of feature vectors with an high dimensionality (e.g., see curvature features of Section 5.1.4) selected from low cardinality datasets.

Random Subspace ensemble (RS) [66, 64, 76] is a valid approach for designing ensembles of classifiers often used in case of datasets affected by the "dimensionality curse", due to the classification performance improvements it offers also in challenging cases. It is based on the *perturbation* of features: each classifier is trained on a training set obtained by reducing the dimensionality of the original dataset by randomly subsampling the features.

Given a collection of N training samples $\mathbf{x_i} \in \mathbb{R}^F$, of K classes $c_k, \quad k = 1, \ldots, K$, Random Subspace randomly selects $M < F$ features without replacement from the original feature space and creates a new training set by projecting each sample into $\mathbb{R}^M$. This procedure is repeated L times where L is the number of final classifiers combined by the sum rule (Eq. 6.10) to formulate the final decision. Note how the single classifiers can be trained on different feature subsets of size M. For the tests of Chapter 7, the ensemble parameters have been set to $L = F/2$, $M = 50$ and support vector machines (SVM) from LibSVM were used as single classifiers.

It is worth noting that the sum rule in this practical case is simpler than the one of Eq. 6.10 [77], and is reported in Eq. 6.11.

$$c = \underset{c_k}{\operatorname{argmax}} \left\{ \frac{1}{K} \sum_{i=1}^{L} P(c_k|\mathbf{x_i}) \right\} \tag{6.11}$$

A great advantage of RS ensembles compared to many other ensemble methods is that they can be coupled to any kind of single classifier and they only need to set two parameters: L, the ensemble size, and M, the size of the feature subsets.

6.3.2 Rotation forest

Rotation Forest [78] is another ensemble of parallel classifiers based on feature sampling in order to tackle the "dimensionality curse" problem described in the previous sections.

Let $\mathcal{D}$ denote a training set of N feature vectors $\mathbf{x_i} \in \mathbb{R}^F, \quad i = 1, \ldots, N$ of F features defined in the feature set $\mathcal{F}$. The method is based, for each classifier $C_j, \quad j = 1, \ldots, L$ in the ensemble of L classifiers, on randomly splitting $\mathcal{F}$ in various subsets and applying Principal Component Analysis (PCA) [79] to only retain the features of $\mathcal{F}$ showing an high information variability in the j-th classifier training set $\mathcal{T}_j \subset \mathcal{D}$. The "rotation" term in the algorithm name comes from the PCA applied to rotate the features and the "forest" comes from the selection of the decision trees as base classifiers for the ensemble.

Let, now, $X \in \mathbb{R}^{N \times F}$ be the dataset $\mathcal{D}$ expressed in matrix form and $\mathbf{Y} = [y_1, \ldots, y_N]^T$ denote the vector of labels $y_i \in \mathcal{C} = \{c_1, \ldots, c_P\}$ with $\mathcal{C}$ the set of the possible classes assignable to each feature vector $\mathbf{x}$.

The construction of the final classifier, formalized in Alg. 6.1, follows these steps:

1. Split $\mathcal{F}$ randomly into K subsets, where K is a parameter of the algorithm. For simplicity, K should be a factor of F, namely each feature subset should contain $M = F/K$ features. Note how the subsets can be either disjoint or intersecting, although the first option is recommended to maximize the chance of high variability among them.

2. Denote by $\mathcal{F}_{i,j}$ the j-th subset of M features for the classifier C_i, with $j = 1, \ldots, K$, and by $X_{i,j}$ the sub matrix of X of the extracted features. For each $\mathcal{F}_{i,j}$, select randomly a non empty subset of classes and then draw a bootstrap sample of objects from $X_{i,j}$ of size 75% of its cardinality. Let $X^B_{i,j}$ denote the bootstrap sample. Run PCA on $X^B_{i,j}$ and store the coefficients of the principal components $a^1_{i,j}, \ldots, a^{M_j}_{i,j}$ of size $M \times 1$, with $M_j \leq M$ as a few eigenvalues may be 0. Note that running the PCA on a subset of classes instead of all of them is done for avoiding to obtain identical coefficients whenever the same feature set happens to be chosen for different classifiers [78].

3. Rearrange the obtained vector of coefficients in a sparse "rotation" matrix $R_i \in \mathbb{R}^{F \times \sum_j M_j}$, as shown in Eq. 6.12.

$$R_i = \begin{bmatrix} a^{(1)}_{i,1}, a^{(2)}_{i,1}, \ldots, a^{(M_1)}_{i,1} & [\mathbf{0}] & \cdots & [\mathbf{0}] \\ [\mathbf{0}] & a^{(1)}_{i,2}, a^{(2)}_{i,2}, \ldots, a^{(M_2)}_{i,2} & ... & [\mathbf{0}] \\ \vdots & \vdots & \ddots & \vdots \\ [\mathbf{0}] & [\mathbf{0}] & \cdots & a^{(1)}_{i,K}, a^{(2)}_{i,K}, \ldots, a^{(M_K)}_{i,K} \end{bmatrix} \tag{6.12}$$

   In order to compute the training set T_i for the classifier C_i, the columns of R_i are first permuted in order to make them correspond to the original features in the full training set X. Denoting by R^a_i the rearranged rotation matrix, the training set for C_i is then computed as $T_i = XR^a_i$.

4. Train each single classifier C_i on the extracted datasets T_i and assign to every feature vector $\mathbf{x} \notin \mathcal{T}$ in exam a class c according to the sum rule (Eq. 6.13).

$$c \leftarrow c_j = \underset{j}{\operatorname{argmax}} \left\{ \frac{1}{L} \sum_{i=1}^{L} P_{i,j}(\mathbf{x}R^a_i) \right\} \tag{6.13}$$

Where $P_{i,j}(\mathbf{x}R_i^a)$ denotes the probability assigned by classifier C_i of the ensemble to the hypothesis $\mathbf{x}$ belongs to class c_j.

Let, now, define an *indicator* function $I(\mathbf{p}) \in \{0, 1\}$ evaluating a predicate $\mathbf{p}$ and returning 1 if the predicate is verified or 0 if not. An alternative sum rule (Eq. 6.14) exploits the indicator function $I(\cdot)$ to avoid the use of the often undisclosed probabilities $P_{i,j}(\mathbf{x}R_i^a)$ in favor of the easier to detect misclassified training vectors [80].

$$c \leftarrow c_j = \underset{j}{\operatorname{argmax}} \left\{ \sum_{i=1}^{L} I(C_i(\mathbf{x}R_i^a) = y_j) \right\} \tag{6.14}$$

Algorithm 6.1 Rotation Forest algorithm

Input:

$X \in \mathbb{R}^{N \times F}$: training set samples arranged as a matrix

$\mathbf{Y} \in \mathcal{C}^{N \times 1}$: label vector for the training set samples

L: the number of classifiers in the ensemble

K: the number of feature subsets for each classifier

Output: The Rotation Forest ensemble classifier

for $i = 1, \ldots, L$ **do**

Split $\mathcal{F}$ into K subsets $\mathcal{F}_{i,j}, j = 1, \ldots, K$

for $j = 1, \ldots, K$ **do**

Extract $X_{i,j}$ from X ▷ The dataset corresponding to the features in $\mathcal{F}_{i,j}$

Eliminate from $X_{i,j}$ a random subset of classes (columns)

Select a bootstrap sample $X_{i,j}^B$ from $X_{i,j}$ of size 75% of $X_{i,j}$

Apply PCA to $X_{i,j}^B$ and collect the coefficients $a_{i,j}^1, \ldots, a_{i,j}^{M_j}$

end for

Build the rotation rotation matrix R_i according to Eq. 6.12

Build R_i^a rearranging the columns of R_i

Train C_i using $(XR_i^a, \mathbf{Y})$ as training set

end for

▷ Classification phase:

Input:

$\mathbf{x} \in \mathbb{R}^F$: feature vector to classify

Output: c: the predicted class for the feature vector $\mathbf{x}$

return $c_j \leftarrow \underset{j}{\operatorname{argmax}} \left\{ \sum_{i=1}^{L} I(C_i(\mathbf{x}R_i^a) = y_j) \right\}$

6.3.3 Adaptive Boosting

Adaptive Boosting [81], also know as "AdaBoost", is a machine learning meta-algorithm that can be used in conjunction with many other types of learning methods to improve their performance. The output of the other learning algorithms, called *weak learners* in this context, is combined into a weighted sum that represents the final output of the boosted classifier.

Differently from Random Subsampling and other parallel ensembles, AdaBoost is an *iterative* training algorithm that at each iteration t adds a weak learner C_t to the boosted classifier ensemble dependent on the classification performance of the previously trained classifiers $C_1, \ldots, C_{t-1}$.

AdaBoost is also *adaptive* in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. Let, in fact, $\mathcal{D} = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ denote a training set of F-dimensional feature vectors $\mathbf{x_i} \in \mathbb{R}^F$ and $h_t : \mathbf{x} \to c \in \{c_1, \ldots, c_K\}$ the classification function of the classifier C_t in the ensemble of T classifiers. In this ensemble method, a set of weights $w_{t,i}$ modeling the feature vector importance and used to sample the training set for C_t are maintained over $\mathcal{D}$. They are initially set to be equal, namely, all training instances have the same importance, while in the subsequent iterations the weights are adjusted so that the weight of the instances misclassified by the previously trained classifiers is increased whereas that of the correctly classified ones is decreased. This expedient allows, step by step, to detect the hard instances and exploit them to train classifiers able to better predict harder instances than the ones predicted by the classifiers trained in the previous steps.

Let, now, consider again the *indicator* function $I(\cdot)$ defined in Section 6.3.2. The misclassification error ε_t for the ensemble at iteration t both depends on the training error of the classifier C_t and the weights $w_{t,i}$ assigned to the samples $\mathbf{x_i} \in \mathcal{D}$ in the previous iteration, as formalized in Eq. 6.15.

$$\varepsilon_t = \sum_{i=1}^{N} I(C_t(\mathbf{x_i}) \neq y_i) w_{t,i} \tag{6.15}$$

Note how the indicator function $I(\cdot)$ is used to only select the weights $w_{t,i}$ associated to the samples erroneously classified by C_t.

ε_t has a fundamental importance, as it is at the base of a factor $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$ that both adjusts the impact of the vote of C_t in the final ensemble and it is used to compute the weights $w_{t+1,i}$ for the next iteration (Eq. 6.16).

$$w_{t+1,i} = \frac{w_{t,i}}{Z_t} \begin{cases} e^{-\alpha_t} & C_t(\mathbf{x_i}) = y_i \\ e^{\alpha_t} & C_t(\mathbf{x_i}) \neq y_i \end{cases} \tag{6.16}$$

where Z_t is a normalization factor such that $\sum_{i=1}^{N} w_{t+1,i} = 1$, namely Z_t is selected to make a distribution over the weights $w_{t+1,i}$.

Once the ensemble has been determined, a given feature vector $\mathbf{x}$ is associated the class c according to the sum rule (Eq. 6.17).

$$c = \underset{c_k}{\operatorname{argmax}} \sum_{t=1}^{T} \alpha_t I(C_t(\mathbf{x}) = c_k) \tag{6.17}$$

Note how the indicator function, this time, for each candidate class c_k only selects the weights α_t for the classifiers C_t that predicted c_k for $\mathbf{x}$. An implementation of AdaBoost is formalized in Alg. 6.2.

Due to the strong dependence of AdaBoost on the weights $w_{t,i}$, the algorithm is highly sensitive to noisy data and outliers, since they corrupt the computed sample distribution at each step.

In some cases, however, AdaBoost may be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing (that is, their error rate is smaller than the 50%), the final model can be proven to converge to a strong learner.

While almost every learning algorithm tends to suit some problem types better than others, and has typically several different parameters and configurations to be adjusted before achieving optimal performance on a given dataset, AdaBoost is often referred to as the best out-of-the-box classifier as it can be used directly without requiring an accurate parameter optimization to reach high classification accuracies.

Finally, several variations of the original AdaBoost approach may be found in literature (e.g, Real AdaBoost, LogitBoost, Gentle AdaBoost), each one usually changing the error function or applying pruning techniques to speed-up the training process. Their treatment, though, is out the scope of this thesis, and the interested reader is invited to consult the relative literature.

Algorithm 6.2 AdaBoost algorithm

Input:

$\mathcal{D} = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$: training set samples

$y_1 \ldots, y_N$: training labels associated to the train samples $\mathbf{x_i}$

$h : \mathbf{x} \to \{c_1, \ldots, c_K\}$: weak learner function

T: number of iterations of the algorithm

$w_{1,i} \leftarrow \frac{1}{N}, \quad i = 1, \ldots, N$ ▷ Weight initialization

for $t \leftarrow 1, \ldots, T$ **do**

Build the training set $\mathcal{T}_t \leftarrow \{\mathbf{x}_1^t, \ldots, \mathbf{x}_N^t\}$ drawing N samples with replacement from $\mathcal{D}$ according to the distribution $w_{t,i}$

Train the classifier C_t on $\mathcal{T}_t$

$\varepsilon_t \leftarrow \sum_{i=1}^{N} I(C_t(\mathbf{x_i}) \neq y_i) w_{t,i}$ ▷ Misclassification error for C_t

if $(\varepsilon_t > 0.5 \vee \varepsilon_t = 0)$ **then**

$T \leftarrow t - 1$

Abort loop

end if

$\alpha_t \leftarrow \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

$w_{t+1,i} \leftarrow \frac{w_{t,i}}{Z_t} \begin{cases} e^{-\alpha_t} & C_t(\mathbf{x_i}) = y_i \\ e^{\alpha_t} & C_t(\mathbf{x_i}) \neq y_i \end{cases}$ ▷ Weight distribution over $\mathcal{D}$ update

end for

return $c \leftarrow \underset{c_k}{\operatorname{argmax}} \sum_{t=1}^{T} \alpha_t I(C_t(\mathbf{x}) = c_k)$ ▷ Predicted class for $\mathbf{x}$

6.3.4 Rotation Boosting

Rotation Boosting [80] is an hybrid classification algorithms based on ensembles which combines the AdaBoost and Rotation Forest approaches, outperforming their single performance in accuracy. Recall than an ensemble of classifiers, in order to achieve a better generalization capability than its constituting members, must be made by extremely accurate classifiers that, at the same time, also disagree as much as possible.

RotBoost algorithm, described in Alg. 6.3, consists in training a set of parallel classifiers using the same approach of Rotation Forest to extract their training sets, with the difference that each classifier is internally optimized in a sequential fashion exploiting AdaBoost method. The selected learning algorithm for the weak classifiers is usually the decision trees, as for AdaBoost and Rotation Forest used singularly.

Algorithm 6.3 RotBoost algorithm

Input: $X \in \mathbb{R}^{N \times F}$: matrix of N training feature vectors of F features
$\mathbf{Y} \in \mathbb{R}^{N}$: vector of the training set labels.
K: number of feature subsets for the Rotation Forest algorithm
S: number of iterations for Rotation Forest
T: number of iterations for AdaBoost

Output: $C_{RB}(\mathbf{x})$: the computed RotBoost classifier

for $s \leftarrow 1, \ldots, S$ **do**
  Compute the rotation matrix R_s^a for the classifier C_s as in Alg. 6.1
  Compute the training set X_s for the classifier C_s as $X_s \leftarrow X R_s^a$
  Initialize the weight distribution over X_s as $w_{s,t}^{(i)} \leftarrow \frac{1}{N}$
  for $t \leftarrow 1, \ldots, T$ **do**
    $\varepsilon_{s,t} \leftarrow \infty$
    while $\varepsilon_{s,t} > 0.5$ **do**
      $w_{s,t}^{(i)} \leftarrow \frac{1}{N}, \quad i = 1, \ldots, N$
      Build the training set $X_{s,t}$ drawing N samples with replacement from X_s according to the distribution $w_{s,t}^{(i)}$
      Train the classifier $C_{s,t}$ on $X_{s,t}$
      $\varepsilon_{s,t} \leftarrow \sum_{i=1}^{N} I(C_{s,t}(\mathbf{x_i}) \neq y_i) w_{s,t}^{(i)}$ ▷ Misclassification error for $C_{s,t}$
    end while
    if $(\varepsilon_t = 0)$ **then**
      $\varepsilon_{s,t} \leftarrow 10^{-10}$
    end if
    $\alpha_{s,t} \leftarrow \frac{1}{2} \ln\left(\frac{1-\varepsilon_{s,t}}{\varepsilon_{s,t}}\right)$
    $w_{s,t+1}^{(i)} \leftarrow \frac{w_{s,t}^{(i)}}{Z_{s,t}} \begin{cases} e^{-\alpha_{s,t}} & C_{s,t}(\mathbf{x_i}) = y_i \\ e^{\alpha_{s,t}} & C_{s,t}(\mathbf{x_i}) \neq y_i \end{cases}$ ▷ Update distribution over X_s
  end for
  $C_s(\mathbf{x}) \leftarrow \underset{c_k}{\operatorname{argmax}} \{\alpha_{s,t} I(C_{s,t}(\mathbf{x}) = c_k)\}$ ▷ Classifier C_s vote
end for
return $C_{RB}(\mathbf{x}) \leftarrow \underset{c_k}{\operatorname{argmax}} \sum_{s=1}^{S} I(C_s(\mathbf{x}) = c_k)$ ▷ Predicted class for $\mathbf{x}$

6.3.5 Random subspace ensemble of RotBoost classifiers

Sections 6.3.1, 6.3.2 and 6.3.3 describe briefly a few classification methods exploiting ensembles of classifiers in order to improve the generalization capabilities of their single components. The hybrid method of Section 6.3.4 further reduces the classification error of new feature vectors leveraging the complementariness of AdaBoost with Rotation Forests.

The current section proposes a new hybrid classification approach, called Random subspace ensemble of RotBoost classifiers [74, 82], integrating in a hierarchical fashion Random Subsampling for the selection of each training set and the aggregation of the opinions of the weak classifiers in the ensemble for taking the final decision, and training each of them with a variation of RotBoost in place of the decision trees employed in the original Random Subsampling algorithm. Differently from RotBoost of Section 6.3.4, the employed implementation replaces PCA with the Neighborhood Preserving Embedding (NPE) feature transform [66, 64] for dimensionality reduction.

Neighborhood Preserving Embedding (NPE)

Neighborhood Preserving Embedding (NPE) [83] is a technique for dimensionality reduction which aims at preserving the local neighborhood structure on the data manifold. It has proven to be more effective than PCA in discovering the underlying nonlinear structure of the data and less sensitive to outliers than other feature transforms.

NPE starts by building a weight matrix to describe the relationships between samples, where each sample is described as a weighted combination of its neighbors. Then, an optimal embedding is selected such that the neighborhood structure is preserved in the reduced space.

Let $\mathbf{x_i} \in \mathbb{R}^F, i = 1, \ldots, N$ denote a generic feature vector of the training set $\mathcal{D}$ represented in a matrix form $X \in \mathbb{R}^{N \times F}$. Let also $\mathcal{G} = (V, A)$ denote a directed graph having feature vectors $\mathbf{x_i}$ as nodes. NPE procedure is based on three main steps:

1. **Construct an adjacency graph using a K-nearest neighbors method** For each node pair $(\mathbf{x_i}, \mathbf{x_j}) \in V \times V$, put a direct edge (arc) from $\mathbf{x_i}$ to $\mathbf{x_j}$ if $\mathbf{x_j}$ is among the K *nearest* neighbors of $\mathbf{x_i}$ according to a given distance metric, as exemplified in Fig. 6.5. K is a parameter of the algorithm and the metric in this case is the Euclidean distance.

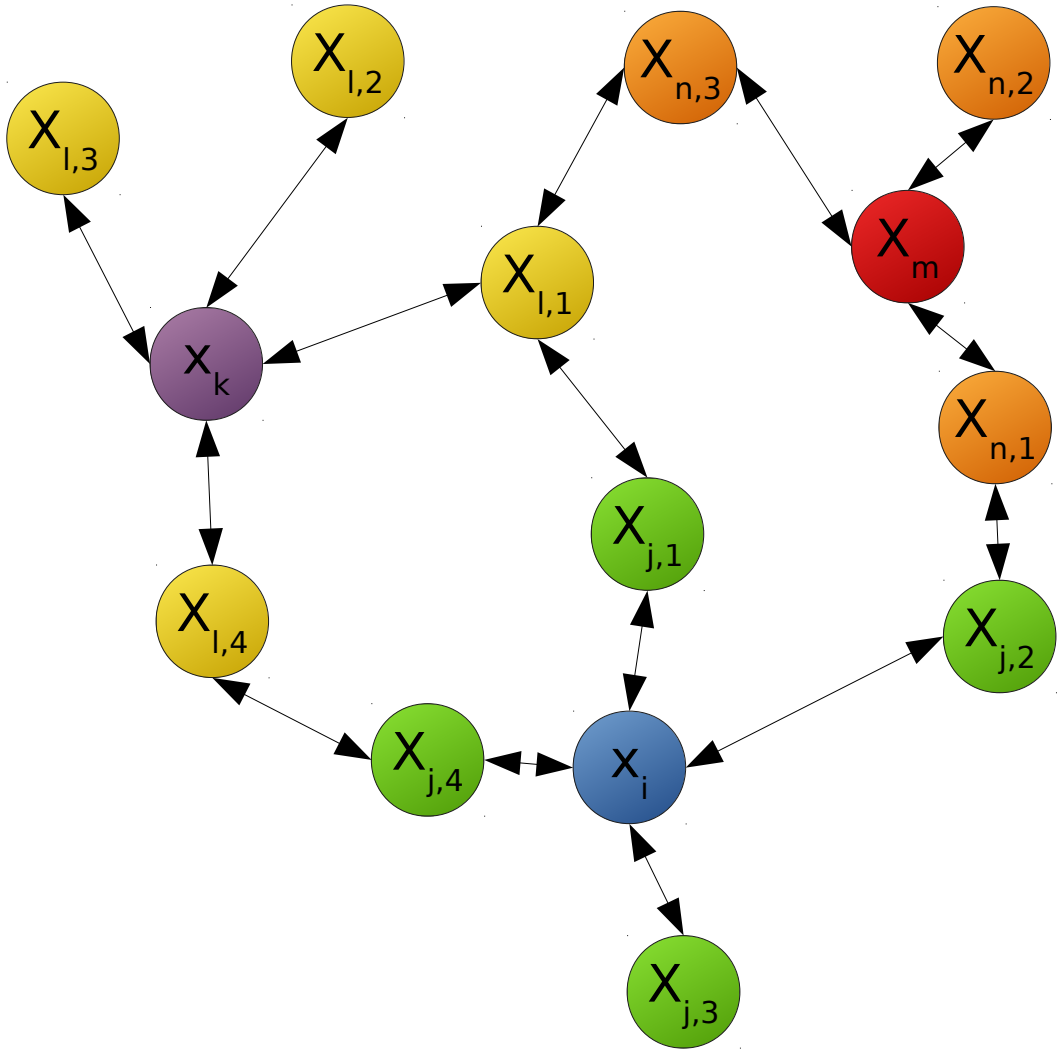


Figure 6.5: Example of adjacency graph for feature vectors in the Euclidean space

2. **Compute the weights of the edges linking the nodes in $\mathcal{G}$**
   Let $W \in \mathbb{R}^{N \times N}$ denote the matrix of weights $w_{i,j}$ of the arcs in $\mathcal{G}$, with $wi, j = 0$ if the arc between $\mathbf{x_i}$ and $\mathbf{x_j}$ does not exist. W is computed by solving the optimization problem of Eq. 6.18.

$$W^* = \underset{W}{\operatorname{argmin}} \sum_{i=1}^{N} \|\mathbf{x_i} - \sum_{j=1}^{N} w_{i,j}\mathbf{x_j}\|^2 \quad \text{subject to} \quad \sum_{j=1}^{N} w_{i,j} = 1 \tag{6.18}$$

3. **Compute the linear projections**
   This step aims at computing the linear transform matrix $A \in \mathbb{R}^{F \times D}$ with $D \leq F$ projecting the feature vectors $\mathbf{x_i} \in \mathbb{R}^F$ in their correspondent vectors $\mathbf{y_i} \in \mathbb{R}^D$ of the dimensionally reduced space, that is $\mathbf{y_i} = A^T \mathbf{x_i}$ for $i = 1, \ldots, N$. The column vectors $\mathbf{a_0}, \mathbf{a_1}, \ldots \mathbf{a_{D-1}}$ of A are the solutions of the generalized eigenvector problem of Eq. 6.19.

$$X(I - W)^T(I - W)X^T\mathbf{a} = \lambda XX^T\mathbf{a} \tag{6.19}$$

   with I identity matrix and λ the eigenvalue associated to the eigenvector $\mathbf{a}$. A is then constructed as $A = [\mathbf{a_0}, \mathbf{a_1}, \ldots \mathbf{a_{D-1}}]$ by rearranging the eigenvectors computed by Eq. 6.19 according to non-decreasing values of their related eigenvalues, that is $\lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_{D-1}$.

Note how NPE has two interesting properties:

- it is fast and suitable for real-time applications, being a linear approach.
- It can be performed in either supervised or unsupervised mode. When the labels of the training patters are available they can be used for building a better weight matrix W.

Moreover, NPE has several advantages over PCA and other dimensionality reduction methods in literature, for example:

- PCA as dimensionality reduction method aims at preserving the global Euclidean structure of the dataset, while NPE aims at preserving the local neighborhood structure and, for this reason, is less sensible to outliers than PCA.
- PCA and other methods for dimensionality reduction do not take into account the actual structure of the underlining datasets. In this way the local relations among the feature vectors are lost during the dimensionality reduction process, while NPE maintains them.

6.4 Feature selection

Classic machine learning based on single classifiers is based on the intuitive idea that the more extracted features of different types, the higher the recognition model accuracy due to a larger amount of information available for the class assignment decision. The high dimensionality of the extracted feature vectors, however, implies heavy computational loads for the feature extraction.

The ensembles of classifiers aim at reducing the computational load by dividing the original classification problem in a sequential or parallel training of single learners on subsets of the initial dataset or subsets of the original feature space, while improving the generalization of the estimated classification model.

Both single classifiers or ensembles, though, unaware of the structure of the underlining training data, often gain only a modest improvement in the classification accuracy over a significant increment of the feature vector length and a consequent higher computational load. The reason of this unfavorable behavior is due to the possible dependencies among the extracted feature.

Highly *correlated* features, in fact, bring no additional information when jointly used, as in front of a change in values of some of them the others change accordingly. Conversely, highly *unncorrelated* features may bring a considerable amount of new information when used together, as the changes of some of them are independent from the changes of the remaining ones. Moreover, the simple juxtaposition of uncorrelated and significant features does not necessarily boost the classification model accuracy, as a limited set of them may already contain all the information needed to determine the class of the performed gesture with the maximum possible accuracy.

For these reasons, this section describes a few feature selection methods in the literature designed to detect the most relevant features of a given dataset and considered in the proposed framework, in order to train a robust classifier with the minimum amount of features leading to the maximum classification accuracy. The following methods have been tested on the features of Chapter 5 for the automatic hand gesture recognition task, leading to interesting interesting results reported in Chapter 7. Note how, while certain feature selection algorithms only depend on the structure of the underlining data disregarding the classification algorithm, others are only defined for specific learners.

6.4.1 Feature selection based on PCA

Principal Component Analysis (PCA) [79] is not properly a feature selection algorithm itself, but a statistical procedure that may be adapted for this purpose.

PCA uses an orthogonal transformation to convert a set of observations of possibly correlated variables (in this case a dataset made of feature vectors) into a set of values of linearly uncorrelated variables called *principal components*. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance, that is, accounts for as much of the variability in the data as possible, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to (namely, uncorrelated with) the preceding components. The principal components are orthogonal because they are the eigenvectors of the covariance matrix, which is symmetric. PCA is sensitive to the relative scaling of the original variables.

While a thorough treatment of PCA is beyond the scope of this thesis and an exhaustive description can be found in [84], note only how PCA is a common dimensionality reduction technique employed in several tasks like compression, face detection, high dimensionality data visualization and machine learning techniques such as Rotation Forests. An example of PCA applied to a bidimensional dataset is shown in Fig. 6.6.

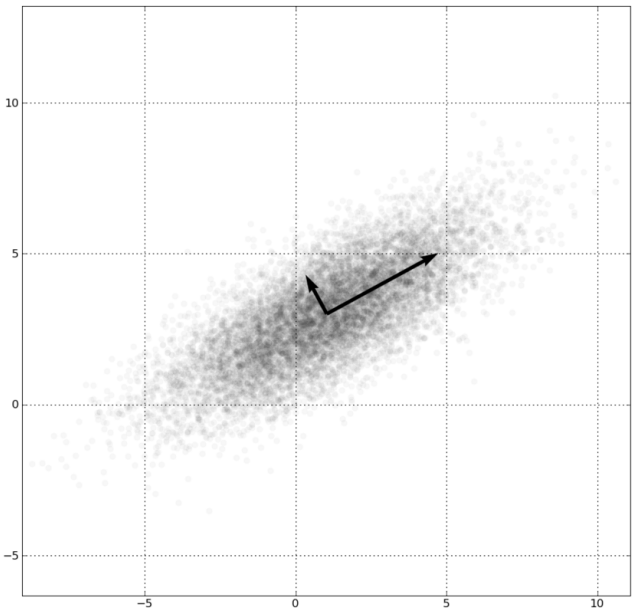


Figure 6.6: Example of PCA on a Gaussian bivariate distribution

The rationale behind PCA for dimensionality reduction and here for feature selection comes from the definition of the transformed feature vectors by the orthogonal transform. Recall, in fact, that PCA may also be visualized as a transformation of the vectors of a multidimensional space (in this case the feature vector space) in vectors of another multidimensional space whose axis (principal components) are orthogonal and ordered by non increasing values of variance. By only selecting the first K principal axis it is possible to reduce the dimensionality of the transformed space limiting the *information loss*, as the variance of the remaining components is minimal, or equivalently, their contribution in "explaining" the data variance is neglectable as the data in the transformed space is almost constant along those axis. In terms of feature importance, the features in the transformed feature vectors are naturally ordered by relevance, as an higher variance corresponds to a more significant contribution in the final decision for the class assignment. According to the Information Theory, in fact, features with high variance encode an higher information than the ones slowly changing.

For these reasons, this work implemented the algorithm based on Principal Component Analysis of Alg. 6.4, which consists in selecting the least number of first features in the transformed space leading to the maximum possible accuracy of the trained model.

6.4.2 Feature selection based on F-score

This feature selection strategy is, analogously to the method of Section 6.4.1, based on the selection on a limited set of features according to their F-score [85], which measures the discrimination of two generic sets of real numbers.

Let $\mathbf{x_i} \in \mathbb{R}^F$ for $i = 1, \ldots, N$ denote, again, a vector of F features of a given training set $\mathcal{T}$. Let us consider, for clarity sake, a binary classifier $C(\mathbf{x}) \in \{-1, 1\}$ assigning 1 to the feature vectors belonging to considered class and -1 otherwise, and suppose to partition $\mathcal{T}$ in $\mathcal{T}^+$ and $\mathcal{T}^- = \mathcal{T} \setminus \mathcal{T}^+$, where $\mathcal{T}^+$ is the subset of the positive instances of the class and $\mathcal{T}^-$ of the negatives denoted by $\mathbf{x_i}^+$ and $\mathbf{x_i}^-$ respectively. In case of multiclass classifier, the method must exploit the one-vs-all or equivalent approaches to reduce the original problem to several binary sub-problems and average the F-scores computed for each of them.

The F-score of the k-th feature, for $k = 1, \ldots, F$, is defined as:

$$F(k) \triangleq \frac{(\overline{\mathbf{x_k}}^+ - \overline{\mathbf{x_k}})^2 + (\overline{\mathbf{x_k}}^- - \overline{\mathbf{x_k}})^2}{\frac{1}{|\mathcal{T}^+|-1} \sum_{j=1}^{|\mathcal{T}^+|} (x_{j,k}^+ - \overline{\mathbf{x_k}}^+)^2 \sum_{j=1}^{|\mathcal{T}^-|} (x_{j,k}^- - \overline{\mathbf{x_k}}^-)^2} \tag{6.20}$$

Algorithm 6.4 Feature selection based on PCA

Input: $X \in \mathbb{R}^{N \times F}$: training set of N feature vectors of F features
$\mathbf{Y}$: label vector for the training samples in X
$C(\mathbf{x})$: selected classifier
T_a: selected accuracy threshold
Output: c: predicted class for a given feature vector $\mathbf{x}$

$P \leftarrow PCA(X)$
$W \leftarrow XP$
$X_{PCA} \leftarrow \mathbf{w_1}$ ▷ Feature selection initialization
for $i \leftarrow 2, \ldots, N$ **do**
Train C with column vectors $\mathbf{w_1}, \ldots, \mathbf{w_i}$ and labels $\mathbf{Y}$ and store the model accuracy a_i (e.g., with leave-one-person-out approach)
if $((a_{i-1} < a_i) \wedge (a_{i-1} > T_a a_i)$ **then**
$X_{PCA} \leftarrow [\mathbf{w_1}, \ldots, \mathbf{w_i}]$
else
Abort loop
end if
end for
$\mathbf{w} \leftarrow \mathbf{x}^T P$ ▷ Transform input vector
$\mathbf{z} \leftarrow [w_1, \ldots, w_K]$ ▷ Reduced feature vector
$c \leftarrow C(\mathbf{z})$ ▷ Predict class for the reduced vector $\mathbf{z}$
return c

where $\overline{\mathbf{x_k}}$, $\overline{\mathbf{x_k}}^+$ and $\overline{\mathbf{x_k}}^-$ denote the averages of the k-th feature values in $\mathcal{T}$, $\mathcal{T}^+$ and $\mathcal{T}^-$ respectively, $|\mathcal{T}|$, $|\mathcal{T}^+|$ and $|\mathcal{T}^-|$ the whole, postive and negative dataset cardinalities, $x^+_{j,k}$ and $x^-_{j,k}$ the k-th feature value in the j-th feature vector of $\mathcal{T}^+$ and $\mathcal{T}^-$ respectively. The numerator indicates the discrimination between the positive and negative sets, and the denominator indicates the discrimination within each of the two sets. The larger the F-score is, the more likely the considered feature is discriminative.

For this reason, analogously to the feature selection algorithm based on PCA (Alg. 6.4), the implemented feature selection method based on F-score in the proposed framework (Alg. 6.5) consists in training a SVM classifier [85] only using the first K features with the highest F-scores until the model accuracy does not improve sensibly by considering more than K features.

Note how F-score, although is one of the fastest and simplest feature selection strategies, has the big disadvantage of not being able to reveal mutual information among features.

Algorithm 6.5 Feature selection based on F-Score

Input: $X \in \mathbb{R}^{N \times F}$: training set of N feature vectors of F features
$\mathbf{Y}$: label vector for the training samples in X
$C(\mathbf{x})$: selected classifier
T_a: selected accuracy threshold
Output: c: predicted class for a given feature vector $\mathbf{x}$

$\mathbf{f} \leftarrow [f_1, \ldots, f_F]$ ▷ Original feature indexes vector
for $k \leftarrow 2, \ldots, F$ **do**
Compute F-Score $F(k)$ for the k-th feature
end for
$z \leftarrow$ permutation of the entries of $\mathbf{f}$ according to decreasing F-Scores
$W \leftarrow$ permutation of the columns of X according to decreasing F-Scores
$\mathbf{s} \leftarrow z_1$ ▷ Feature selection initialization
$X_F \leftarrow \mathbf{w_1}$
for $k \leftarrow 1, \ldots, F$ **do**
Train C with column vectors $\mathbf{w_1}, \ldots, \mathbf{w_k}$ and labels $\mathbf{Y}$ and store the model accuracy a_k (e.g., with leave-one-person-out approach)
if $((a_{k-1} < a_k) \wedge (a_{k-1} > T_a a_k)$ **then**
$X_F \leftarrow [\mathbf{w_1}, \ldots, \mathbf{w_k}]$
$\mathbf{s} \leftarrow [z_1, \ldots, z_k]$
else
Abort loop
end if
end for

$\mathbf{z} \leftarrow$ vector of $\mathbf{x}$ entries selected by $\mathbf{s}$ ▷ Transform input vector
$c \leftarrow C(\mathbf{z})$ ▷ Predict class for the reduced vector $\mathbf{z}$
return c

6.4.3 Feature selection based on Random Forests

This feature selection scheme, analogously to the algorithms of Sections 6.4.1 and 6.4.2, assigns a score to each feature according to its relevance and selects the first relevant K features maximizing the classification model accuracy.

In this case the full training set $\mathcal{D}$ is firstly trained with a Random Forest classifier [26] and the Out-of-Bag error (Section 6.2) estimated. Then, in order to measure the importance of the various features, the values of one of the features in the dataset are permuted and the Out-of-Bag error is estimated again. The procedure is repeated for each feature and the importance of each feature is given by the normalized average increase of the Out-of-Bag error after the permutation. A more detailed description can be found in [86].

6.4.4 Sequential feature selection

Sequential feature selection algorithm [87] is an effective selection scheme based on iteratively constructing the minimal set of most relevant features maximizing the accuracy of the classification model. It requires the definition of a *search strategy* in the feature research space of size $O(2^F)$ with F the number of features, and of a performance metric on the evaluated possible solution $S \in \{0,1\}^F$, where S is a binary string representing the selected feature subset.

There are mainly two search strategies in literature:

- **Forward Sequential Selection (FSS):** starting from the empty set, the (greedy) algorithm first evaluates all the possible subsets of 1 feature and selects the one that maximizes the recognition accuracy of the classifier trained on it. In the further steps, the algorithm extends incrementally the feature set with one feature following the same rationale, until adding new features does not lead to a relevant performance improvement.

- **Backward Sequential Selection (BSS):** analogously to FSS, the algorithm this time starts from the full feature set and removes one feature per round until the classification performance does not drop sensibly.

Forward Sequential Selection coupled with SVM as classifier, formalized in Alg. 6.6, has been evaluated in the experiments of Chapter 7 for the automatic gesture recognition task. FSS is usable both for classification and for regression. In the first case, the performance metric is usually the number of misclassified feature vectors, while in the latter case it is usually the sum of squared errors of the predictions.

Algorithm 6.6 Forward Sequential Selection algorithm

▷ Training phase:

Input:

$\mathcal{F} = \{f_1, \ldots, f_F\}$: the complete feature set

$\mathcal{D} = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$: dataset of N feature vectors of length F

$y_1, \ldots, y_N$: labels associated to the feature vectors in $\mathcal{D}$

$C(\mathbf{x})$: selected classifier

T_a: selected accuracy threshold

Output: c: predicted class for a given feature vector $\mathbf{x}$

$\mathcal{S} \leftarrow \phi$ ▷ Selected feature subset initialization

$\mathcal{R} \leftarrow \mathcal{F}$ ▷ Set of the not selected features

$acc \leftarrow 0$ ▷ Starting accuracy initialization

$acc^* \leftarrow 0$

repeat

$acc \leftarrow acc^*$

for $i \leftarrow 1, \ldots, |\mathcal{R}|$ **do**

$\mathcal{S}_i \leftarrow \mathcal{S} \cup \{f_i^{\mathcal{R}}\}$ with $f_i^{\mathcal{R}}$ the i-th feature contained in $\mathcal{R}$

$\mathcal{T} \leftarrow \mathcal{D}$ only selecting the features indicated in $\mathcal{S}_i$

Train C on $\mathcal{T}$ and store the model accuracy acc_i (e.g., with leave-one-person-out approach)

end for

$acc^* \leftarrow \max_i acc_i$

$\mathcal{S} \leftarrow \operatorname{argmax}_{\mathcal{S}_i} acc_i$

until $acc^* > T_a acc$

▷ Prediction phase:

$\mathbf{z} \leftarrow$ vector of entries of $\mathbf{x}$ selected by $\mathcal{S}$ ▷ Transform input vector

$c \leftarrow C(\mathbf{z})$ ▷ Predict class for the reduced vector $\mathbf{z}$

return c

6.5 Classification performance

A final important aspect of the proposed automatic gesture recognition approach, and more generally of every computer vision task, is the evaluation of the performance of the recognition model computed with the selected classifier.

The easiest and most intuitive metric for assessing the model quality is often the estimated classification accuracy, e.g. the k-fold cross validation on the full dataset or the ratio of the correctly classified feature vectors of the test set over the test set cardinality. The main idea in this case consists in selecting the classification method which outperforms the others on the given training set, usually after an accurate optimization of the classifier parameters. This rationale, however, may lead to wrong assumptions as the high estimated classification accuracy is more likely to be due to the model overfitting on the limited dataset. The actual capability of the model of correctly classifying new feature vectors could be, instead, much lower than the expected one.

In order to avoid biased evaluations, literature offers several classification performance metrics able to assess the actual classification capabilities of the trained model. This section describes three measures used to test the performance of the classifiers employed for the automatic hand gesture recognition problem in this thesis.

6.5.1 Area under the Receiver Operating Characteristic curve (AUC)

The first metric, originally deployed for binary classifiers and then extended for the multiclass case, is based on the Receiver Operating Characteristic (ROC) [88], or ROC curve. ROC is a graphical plot that illustrates the performance of a binary classifier system varying its parameters.

The curve is created by plotting the *true positive* rate against the *false positive* rate for different configurations of the classifier parameters. The true positive rate is also known as *sensitivity* in biomedicine, or *recall* in machine learning. The false positive rate is also known as the *fall-out* and can be calculated as (1 - specificity). The ROC curve is then the sensitivity as a function of fall-out. Generally, if the probability distributions for both detection and false positive are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from $-\infty$ to $+\infty$ of the detection probability in the y-axis versus the cumulative distribution function of the false-positive probability in x-axis).

Fig. 6.8 compares the predictive powers of a few classifiers in the ROC space for determined settings.

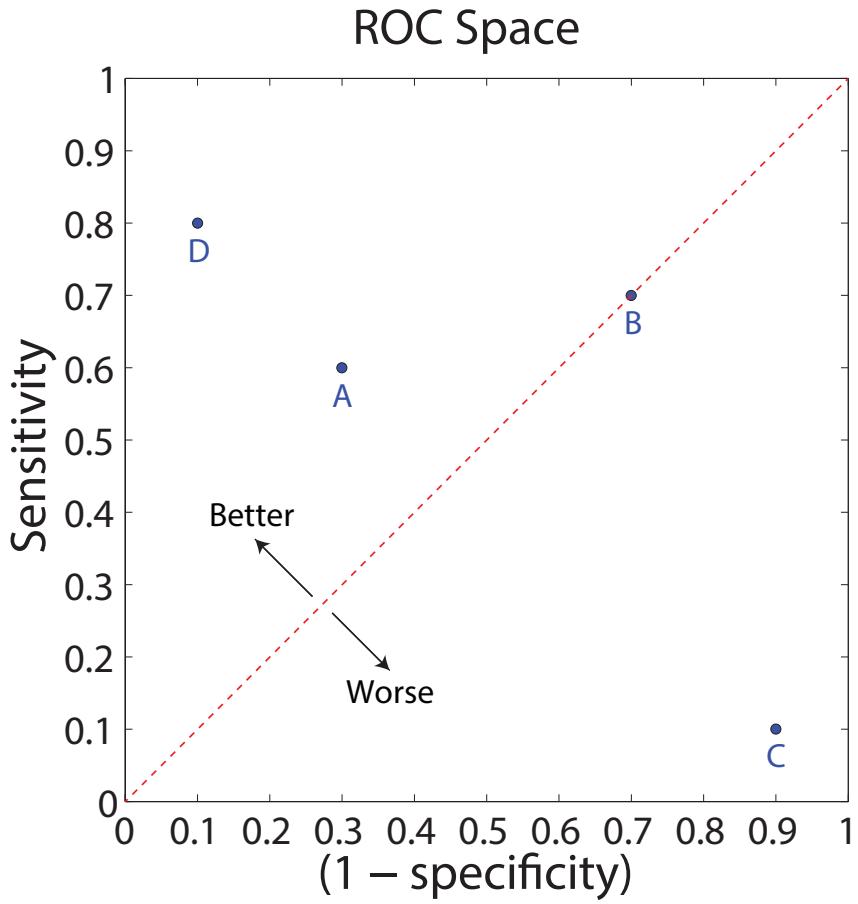


Figure 6.7: Comparison of the predictive power of different classifiers in the ROC space: good (A), random guess (B), poor (C), best (D)

ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. In particular, the area under the ROC curve (AUC), can be interpreted as the probability that the classifier will assign an higher score to a randomly chosen positive sample than it would to a randomly chosen negative one, assuming that higher scores are referred to the samples belonging to a given class. Namely, AUC is a measure of the classifier *discrimination*, that is its capability of correctly classifying a random pair of feature vectors where the one is a positive sample and the other not.

In the multi-class problem, AUC is calculated using the one-versus-all approach [70]: a given class is considered as "positive" and all the other classes are considered as "negative") and the average AUC is reported.

6.5.2 Wilcoxon Signed-Rank Test

Wilcoxon signed-rank test [89] is an algorithm for measuring the differences in performance of two classifiers $C_1(\mathbf{x})$ and $C_2(\mathbf{x})$ on the same dataset $\mathcal{D}$ of N feature vectors, or a set of N different datasets $\mathcal{D}_i, i = 1, \ldots, N$, to assess which classifier better performs.

The test ranks the differences in performance or *score* of two classifiers for each sample or data set, ignoring the signs, and compares the ranks for the positive and the negative differences. Let d_i denote the difference between the scores of the two classifiers on the i-th sample or dataset. The differences are ranked according to their absolute values, and average ranks are assigned in case of ties, as exemplified in Table 6.1 which compares the accuracy of the C4.5 classification algorithm [90] on different datasets for two parametrizations.

	C4.5 (unoptimized)	C4.5 (optimized)	Difference	Rank
adult (sample)	0.763	0.768	+0.005	3.5
breast cancer	0.599	0.591	0.008	7
breast cancer wisconsin	0.954	0.971	+0.017	9
cmc	0.628	0.661	+0.033	12
ionosphere	0.882	0.888	+0.006	5
iris	0.936	0.931	0.005	3.5
liver disorders	0.661	0.668	+0.007	6
lung cancer	0.583	0.583	0.000	1.5
lymphography	0.775	0.838	+0.063	14
mushroom	1.000	1.000	0.000	1.5
primary tumor	0.940	0.962	+0.022	11
rheum	0.619	0.666	+0.047	13
voting	0.972	0.981	+0.009	8
wine	0.957	0.978	+0.021	10

Table 6.1: Example of Wilcoxon Signed-Ranks Test ranks

Let, now, R^+ denote the sum of ranks for the data sets on which the second algorithm outperformed the first, and R the sum of ranks for the opposite, as reported in Eq. 6.21. The ranks of $d_i = 0$ are split evenly among the sums, ignoring one d_i in case of an odd number of them.

$$
\begin{aligned}
R^+ &= \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \\
R^- &= \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)
\end{aligned}
\tag{6.21}
$$

Let also T denote the smallest sum between R^+ and R^-, that is $T = \min\{R^+, R\}$. T, along with N and the confidence value α (often $\alpha = 0.005$), determines whether the *null hypothesis*, that is the hypothesis the two classifiers perform equally well, can be considered valid or not. Note how the first critical values for T are pre-computed in Table 6.2, also reported in several books of statistics.

N	$\alpha = 0.05$	$\alpha = 0.01$	N	$\alpha = 0.05$	$\alpha = 0.01$
5	–	–	18	40	27
6	0	–	19	46	32
7	2	–	20	52	37
8	3	0	21	58	42
9	5	1	22	65	48
10	8	3	23	73	54
11	10	5	24	81	61
12	13	7	25	89	68
13	17	9	26	98	75
14	21	12	27	107	83
15	25	15	28	116	91
16	29	19	29	126	100
17	34	23	30	137	109

Table 6.2: Wilcoxon critic values look-up-table

According to this test, the two classifiers have actual different performance only if $T(\alpha, N) < T_{crit}(\alpha, N)$ with T_{crit} critical value for T in case of $N \leq 25$ with a confidence value of α.

For example, Table 6.1 compares the performance of two classifiers on $N = 14$ datasets, where the first classifier parameters have not been optimized while the second one have been fully optimized. $R^+ = 93$ and $R^- = 12$, so $T = 12$. According to Table 6.2, for $N = 14$ and $\alpha = 0.05$ the null hypothesis can be discarded only if $T < T_{crit} = 21$. Since $T = 12$, the fully optimized classifier actually outperforms its unoptimized version.

For an higher number of datasets or samples, the test requires the computation of the *z-score* (from the Normal distribution, called "Z-Distribution" in statistics) with Eq. 6.22.

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{6.22}$$

In this case, for a two-tailed distribution (as the test is only interested in the absolute difference of the classifier scores) the null hypothesis can be only discarded if the area under the normal distribution for the computed z-score in Eq. 6.22 is lower than the area under the curve for the selected α. Recall that the area under the normal curve represents the probability that the difference in performance between the two classifiers only happen by chance.

For example, $\alpha = 0.05$ corresponds to an area under the normal curve of $\alpha/2 = 0.025$ as we are considering a two-tailed distribution, which in turn corresponds to a z-score of -1.96. It follows that, for $\alpha = 0.05$, the null-hypothesis can be only rejected if z is smaller than 1.96, as it would mean that the differences in performance are most likely to happen not by chance than the given threshold (in this case of the 95%). Since for $T = 12$ and $N = 14$ the computed z-score in Eq. 6.22 is $z = -2.5424 < -1.96$, which corresponds to an area under the normal curve of $0.0055 < 0.025$, that is the 0.55% of probability the differences in performance happend by chance (or equivalently the probability of the 99.45% they are reliable), the test states again that the optimized classifier actually outperforms its unoptimized version.

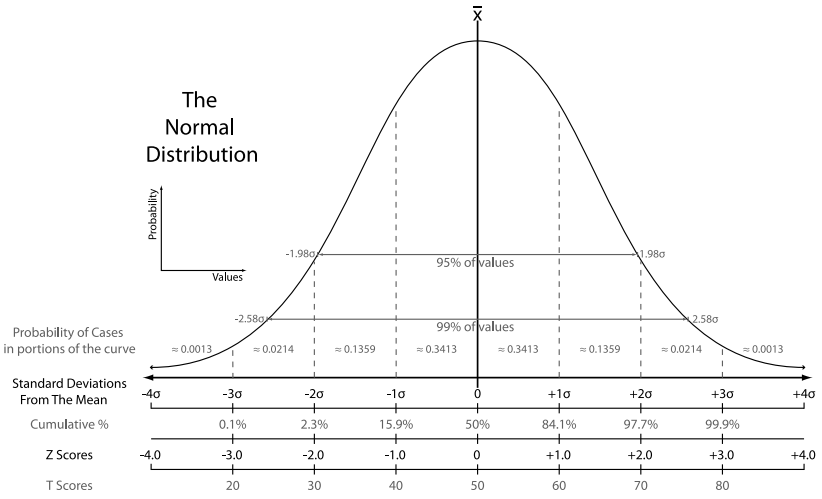


Figure 6.8: Normal distribution

6.5.3 Q statistics

Q statistics is a statistical metric used to quantify the diversity between pairs of classifiers by their classification outcomes. In particular, this measure was used in certain tests in Chapter 7 to assess the presence of a sufficient degree of statistical independence between pairs of feature sets extracted from the same dataset, a necessary condition for motivating their combination (or the combination of different classifiers in the same ensemble) in order to improve the overall recognition performance.

Let $\mathcal{D} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$ be a labeled data set of F-dimensional feature vectors $\mathbf{x_i} \in \mathbb{R}^F$ with labels $y_i \in \{c_1, \ldots, c_K\}$ associated to K classes. It is possible to represent the output of a classifier C_j on $\mathcal{D}$ as a N-dimensional binary vector $\mathbf{s_j} = [s_{1,j}, \ldots, s_{N,j}]^T$, where $s_{i,j} = 1$ if C_j classified correctly $\mathbf{x_i}$ and $s_{i,j} = 0$ if not.

Given two classifiers C_i and C_j, the Q statistic for C_i and C_j is defined as:

$$Q_{i,j} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{6.23}$$

where N^{ab} denotes the number of feature vectors $\mathbf{x_l}$ of $\mathcal{D}$ for which $s_{l,i} = a$ and $s_{l,j} = b$, e.g., N^{10} denotes the number of feature vectors correctly classified by C_i but misclassified by C_j (Tab. 6.3).

$Q_{i,j}$ assumes values within the range $[-1, 1]$ and, for statistically independent classifiers, $Q_{i,j}$ converges to 0. Classifiers that tend to recognize the same feature vectors correctly have positives values of Q, while those which classifies erroneously different feature vectors make Q negative.

	C_j correct (1)	C_j wrong (0)
C_i correct (1)	N^{11}	N^{10}
C_i wrong (0)	N^{01}	N^{00}
Total:	$N = N^{00} + N^{01} + N^{10} + N^{11}$	

Table 6.3: Relationship between a pair of classifiers

For an ensemble of L classifiers, Eq. 6.23 is replaced by Eq. 6.24 computing the average Q statistic over all the pairs of classifiers.

$$Q_{avg} = \frac{2}{L(L-1)} \sum_{i=1}^{L-1} \sum_{j=i+1}^{L} Q_{i,j} \tag{6.24}$$

Chapter 7

Results

This Chapter discusses the performance of the proposed automatic hand gesture recognition framework on the basis of the results gathered from several tests performed on three selected datasets, each one representing a different "gesture dictionary" made by a subset of gestures from the American Sign Language.

- **MICROSOFT dataset**: provided by Microsoft [17], contains $R = 10$ repetitions of $G = 10$ gestures performed by $P = 10$ different people for a total of $N = 1000$ frames acquired with a Microsoft Kinect (ver. 1) range camera. A representative picture for each gesture is shown in Fig. 7.1. The dataset provides both the RGB image and the associated depth map for all the frames, although the calibration data is missing.

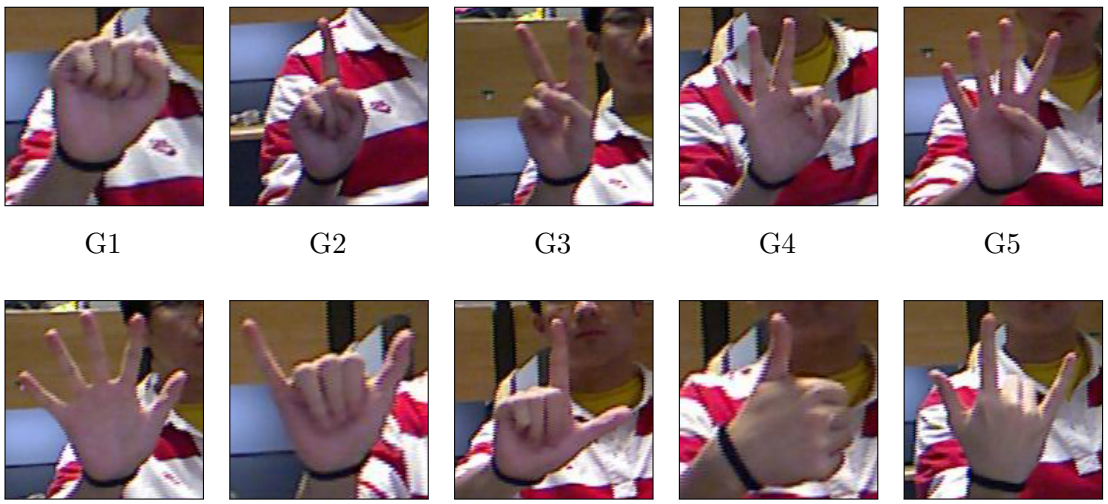


Figure 7.1: Gestures of MICROSOFT dataset

- **LTTM dataset**: acquired in the Multimedia Technology and Telecommunications Lab (LTTM) of the University of Padova with a Microsoft Kinect (ver. 1) range camera, contains $R = 10$ repetitions of $G = 12$ different gestures performed by $P = 14$ different people for a total of $N = 1680$ frames. A representative picture for each gesture is shown in Fig. 7.2 while the complete dataset is made available at the url `http://lttm.dei.unipd.it/paper_data/gesture/`, and provides both the RGB image and the aligned depth map for all the frames.

Figure 7.2: Gestures of LTTM dataset

- **LEAPNECT dataset**: acquired in the Multimedia Technology and Telecommunications Lab (LTTM) of the University of Padova with the hybrid setup depicted in Fig. 2.21, mabe by a Microsoft Kinect (ver. 1) range camera and a Leap Motion, contains $R = 10$ repetitions of $G = 10$ gestures performed by $P = 14$ people for a total of $N = 1400$ different frames. A representative picture for each gesture is shown in Fig. 7.3 while the

complete dataset is made available at the url `http://lttm.dei.unipd.it/downloads/gesture/`, and provides the RGB image, the aligned depth map and the Leap Motion data for all the frames. Note how the two devices have been jointly calibrated using the approach of Section 5.3.1 and synchronized in time. A software synchronization has been used, since its precision was sufficient for the recognition of gestures based on static poses. For gesture based on fast movements, instead, probably a more accurate synchronization approach would be needed.

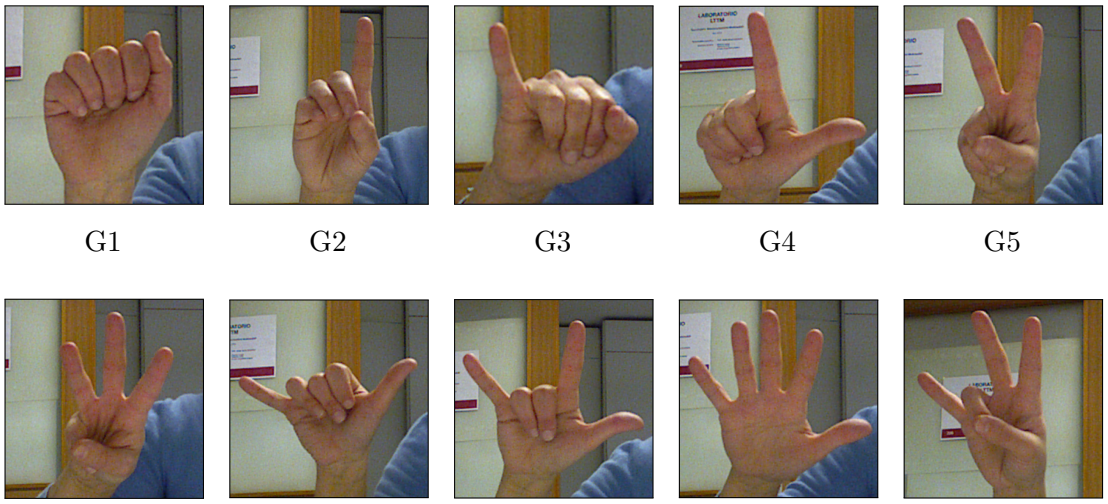


Figure 7.3: Gestures of LEAPNECT dataset

It is worth noting that all the datasets described above only contain *static* gestures used for automatic sign language interpretation purposes, in order to compare the proposed gesture recognition algorithm with the other approaches in literature. Dynamic gesture recognition will probably be a future extension of the work in this thesis.

The remainder of this chapter is articulated as follows: Sections 7.1 and 7.2 compare respectively the performance of single learners and ensembles of classifiers on the features of Chapter 5 extracted from the considered datasets. Section 7.3 evaluates the dimensionality reduction possibility with the feature selection methods of Section 6.4, and finally Section 7.4 measures the execution times of the main steps of proposed gesture recognition algorithm running on a real system.

7.1 Single classifier performance

This section analyzes the classification performance of the various feature sets described in Chapter 5 and extracted from the MICROSOFT, LTTM and LEAP-NECT datasets.

Let, for clarity sake, recall the notation of the considered depth feature vectors of Chapter 5 in this chapter:

- $\mathbf{F_1^l}$: hand contour distances from the palm center (plot alignment version).
- $\mathbf{F_2^l}$: hand contour distances from the palm center (mask alignment version).
- $\mathbf{F^e}$: hand contour distances from the palm plane.
- $\mathbf{F_1^z}$: hand contour similarities with ZNCC (plot alignment version).
- $\mathbf{F_2^z}$: hand contour similarities with ZNCC (mask alignment version).
- $\mathbf{F_1^s}$: hand contour similarities with SSD (plot alignment version).
- $\mathbf{F_2^s}$: hand contour similarities with SSD (mask alignment version).
- $\mathbf{F^c}$: hand contour curvatures.
- $\mathbf{F^a}$: palm morphology features.
- $\mathbf{F^{cp}}$: convex hull perimeters ratio.
- $\mathbf{F^{ca}}$: convex hull areas ratio.
- $\mathbf{F^{cc}}$: convex hull connected components areas ratio.

and the notation of the Leap Motion features of interest:

- $\mathbf{F_L^\theta}$: fingertip orientations.
- $\mathbf{F_L^d}$: fingertip distances from the palm center.
- $\mathbf{F_L^e}$: fingertip distances from the palm plane.
- $\mathbf{F_L^p}$: fingertip positions.

For each gesture, one of the repetitions in the training sets was used for the distance plot template computation (Eq. 5.8), required for the extraction of several feature sets.

Table 7.1 reports the estimated recognition accuracies obtained with a single SVM classifier, implemented in the OpenCV library, trained on the datasets described in the chapter introduction. As stated in Section 6.1, the proposed framework exploited a radial kernel optimized with the leave-one-person-out approach. The combination of multiple descriptors was tackled by simply juxtaposing the feature vectors belonging to the respective feature sets.

	Estimated accuracy (%)		
Feature set	MICROSOFT	LTTM	LEAPNECT
$\mathbf{F_1^l}$	85.4	68.4	- (-)
$\mathbf{F_2^l}$	-	-	91.9 (94.4)
$\mathbf{F^e}$	56.1	46	- (-)
$\mathbf{F_1^z}$	87.4	60.4	56.8 (57.1)
$\mathbf{F_2^z}$	-	-	68.5 (68.7)
$\mathbf{F_1^s}$	49.5	37.8	31.9 (29.5)
$\mathbf{F^c}$	91.1	89.5	87.3 (86.2)
$\mathbf{F^a}$	61.5	45.4	- (-)
$\mathbf{F^{cp}}$	-	37	52.9 (52.9)
$\mathbf{F^{ca}}$	-	29.6	29 (30.6)
$\mathbf{F^{cc}}$	-	72	70.5 (71.1)
$\mathbf{F_1^l + F^c}$	93.4	91.4	- (-)
$\mathbf{F_2^l + F^c}$	-	-	92.8 (96.4)
$\mathbf{F_1^z + F_1^s}$	92.1	78.3	- (-)
$\mathbf{F_1^z + F^c}$	98.5	91.6	89.7 (-)
$\mathbf{F^{cp} + F^{ca}}$	-	51.2	64 (64.4)
$\mathbf{F_1^l + F^e + F^c}$	94.7	93.6	- (-)
$\mathbf{F_1^l + F^a + F^c}$	95.2	92	- (-)
$\mathbf{F^{ca} + F^{cp} + F^{cc}}$	-	73.2	79.1 (81.3)
$\mathbf{F_1^l + F^e + F^a + F^c}$	96.4	93.5	- (-)
$\mathbf{F_1^z + F_1^s + F^{cp} + F^{ca} + F^{cc}}$	-	86.8	- (-)

Table 7.1: Comparison of the depth features accuracies for three datasets

where for LEAPNECT dataset the values inside parentheses are referred to the case of joint usage of a range camera with the Leap Motion [57], while the others are referred to the case of usage of the two devices independently [58].

A first consideration on Table 7.1 is that the results of the tests on MICROSOFT dataset are generally much better than the test outcomes on the remaining datasets. This is an expected result, since MICROSOFT dataset is made of a limited number of gestures performed by people of the same ethnic group and age range, thus sharing several physical characteristics, and favorable lighting and framing conditions (good illumination and camera framing the user's hand frontally). MICROSOFT dataset is, then, rather homogeneous.

LTTM and LEAPNECT datasets are, instead, more challenging both because account for users of different age, sex, physique and ethnic group, and for the acquisition constraints in certain cases. For example, LEAPNECT acquisition constrained the hand to move within the limited Leap Motion operating range preventing the range camera from acquiring the full hand frontally. Moreover, LEAPNECT dataset contain an higer number of gestures and both LTTM and LEAPNECT datasets purposely include gestures known to raise several recognition ambiguities, in order to test the robustness of the proposed gesture recognition algorithm. Note how, for example, G3, G6 and G9 of LTTM dataset only show a raised finger, that in G3 and G9 is the same finger completely raised in the first case and half bent in the latter one.

Hand contour distances from the palm center ($\mathbf{F_1^l}$) alone provide an accuracy of almost 90% in MICROSOFT dataset and 70% in LTTM one, meaning the descriptor is able to discriminate most of the performed gestures without requiring further information. As already stated in Section 5.1.1, this feature set is very good in capturing the fact that the various fingers are folded over the palm or raised, an important element in the recognition of several gestures. While this is true for MICROSOFT and LEAPNECT datasets, where the gestures are strongly characterized by their hand contours, the accuracy drop in LTTM dataset is due to a considerable number of gestures sharing the same number of raised fingers in similar configurations, whose differences are not well captured by this descriptor.

Fig. 7.4 compares the confusion matrices for MICROSOFT and LTTM datasets, where the cells in yellow denote the true positives and the ones with a different color the false positives. In particular, the cells in gray account for percentages of false positives below the 5%, cells in orange the ones within the range $[5\%, 10\%)$ and the cells in red percentages of classification failures above 10%. Fig. 7.4(a) shows that the descriptor is perfectly able to discriminate the close fist gesture

(G1) from the the other gestures with one or more raised fingers in MICROSOFT dataset, while it fails sometimes in discriminating gestures containing subsets of the same raised fingers. For example, G4 is often misclassified as G5 and G5 as G6, which form a kind of sequence in the dataset. This is probably due to slight misalignment errors of the distance plots or to larger fingers falling between consequent regions in the gesture templates. The same behavior is shown in Fig. 7.4(b), e.g. with G7 and G8 or G3 and G5 of LTTM dataset.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	1	0	0	0	0	0	0	0	0	0
G2	0	0.87	0	0.03	0	0	0.02	0.08	0	0
G3	0	0.02	0.84	0.02	0.11	0.01	0	0	0	0
G4	0	0	0.09	0.78	0.11	0.01	0	0	0.01	0
G5	0	0	0.02	0.08	0.7	0.19	0	0	0	0.01
G6	0	0	0.02	0	0.05	0.93	0	0	0	0
G7	0	0.01	0	0	0	0	0.81	0.01	0.17	0
G8	0	0	0.02	0.06	0.01	0	0.02	0.82	0	0.07
G9	0.02	0	0	0	0	0	0.06	0.02	0.9	0
G10	0	0	0	0	0.02	0.05	0	0.03	0.01	0.89

(a) MICROSOFT dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.91	0.07	0	0	0	0.01	0	0	0.01	0	0	0
G2	0.13	0.74	0	0.07	0.01	0.01	0	0	0.03	0	0	0.01
G3	0	0	0.68	0	0.24	0.01	0.01	0.01	0.04	0.01	0	0
G4	0	0.04	0	0.61	0.01	0.01	0.01	0.04	0.14	0	0.01	0.14
G5	0	0	0.17	0.01	0.59	0.03	0.04	0.03	0.09	0.03	0	0.01
G6	0.04	0.04	0.01	0	0.02	0.68	0	0.01	0.15	0.04	0.01	0
G7	0	0	0.01	0.03	0.14	0	0.41	0.41	0	0	0	0.01
G8	0	0.01	0	0.1	0.03	0	0.28	0.56	0	0	0.01	0.02
G9	0.04	0.05	0.11	0.06	0.09	0.2	0.01	0	0.44	0	0	0
G10	0.01	0.01	0.03	0	0.01	0.1	0	0	0.01	0.81	0.01	0.01
G11	0	0.01	0	0.02	0	0.01	0	0.01	0	0.01	0.93	0.01
G12	0	0.02	0	0.06	0.01	0	0.02	0.01	0	0	0.01	0.86

(b) LTTM dataset

Figure 7.4: Distances from the palm center (plot alignment version)

The newest version of the distance descriptor ($\mathbf{F}_2^l$) is the most performing feature set that alone is able to recognize almost all the gestures in LEAPNECT dataset. This is both due to an higher number of features respect to the previous version (180 VS 24 or more), which carry more information, and to the fact that

the hand outline computed from the depth mask is more accurate than the one estimated from the 3D finger samples only.

Hand contour distances from the palm plane are, instead, a not very performant feature set in general (reporting an accuracy lower than the 60% on overall) and do not allow to discriminate alone the performed gestures with a sufficient degree of accuracy (see Fig. 7.5). This is both due to the fact that in most gestures in the selected datasets the fingers are either raised or lay very close to the palm plane, thus not allowing the descriptor to leverage its real capabilities, and to the varying accuracy in the plane fitting of Alg. 4.3. Moreover, since the range cameras do not return a volumetric description of the framed scene but only its surface geometry, when the fingers occlude a considerable part of the palm region the plane is more likely to be fitted on the finger samples than on the palm ones.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.77	0	0	0.01	0.11	0.02	0.07	0.01	0.01	0
G2	0	0.62	0.06	0	0.01	0	0.07	0.18	0	0.06
G3	0	0.08	0.84	0	0.01	0	0	0.05	0	0.02
G4	0.01	0	0	0.63	0.28	0.04	0.01	0.01	0	0.02
G5	0.22	0.01	0.02	0.19	0.33	0.12	0	0.04	0	0.07
G6	0.24	0	0	0.11	0.28	0.21	0.03	0.03	0.01	0.09
G7	0.26	0.01	0	0.03	0.03	0	0.52	0.01	0.05	0.09
G8	0.01	0.06	0.09	0.03	0.08	0.03	0.08	0.42	0.01	0.19
G9	0.05	0	0	0	0	0.05	0.08	0.03	0.63	0.16
G10	0.01	0.01	0	0	0	0.06	0.08	0.09	0.11	0.64

(a) MICROSOFT dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.61	0.01	0	0.01	0	0.23	0	0	0	0.06	0.02	0.06
G2	0	0.81	0	0.08	0.01	0.01	0	0.01	0	0.03	0.04	0.01
G3	0	0.01	0.46	0	0.06	0.01	0.18	0.04	0.22	0.01	0.01	0
G4	0.01	0.06	0	0.68	0	0.01	0	0.04	0.01	0.01	0.04	0.14
G5	0.02	0.05	0.08	0.01	0.35	0.01	0.21	0.06	0.14	0.05	0.01	0
G6	0.5	0.05	0	0.01	0.01	0.27	0	0.01	0.01	0.09	0.01	0.04
G7	0	0.01	0.11	0	0.14	0	0.47	0.1	0.12	0.02	0.01	0.01
G8	0.01	0.06	0.04	0.07	0.16	0.01	0.2	0.16	0.07	0.02	0.09	0.11
G9	0	0.01	0.17	0	0.17	0.01	0.18	0.01	0.42	0.01	0.01	0.01
G10	0.16	0.03	0.04	0.03	0.02	0.12	0	0	0.02	0.45	0.11	0.01
G11	0.09	0.05	0.02	0.13	0.01	0.03	0.01	0.02	0.06	0.1	0.37	0.11
G12	0.13	0.04	0	0.12	0	0.18	0.01	0.01	0	0.01	0.04	0.46

(b) LTTM dataset

Figure 7.5: Distances from the palm plane

Palm morphology features ($\mathbf{F^a}$), along with distances from the palm plane ones report, again, a ratherly low accuracy in LTTM dataset (45.4%) and better results in MICROSOFT dataset (61.5%). Fig. 7.6(a) shows that the descriptor discriminates G4 of MICROSOFT dataset slightly better than the hand contour distances from the palm center and the improvement respect to the distances from the palm plane is even more evident, due to the fact that the palm morphology features capture well the thumb and index fingers forming a ring almost orthogonal to the palm plane. A different behavior is shown in Fig. 7.6(b) for the LTTM dataset, probably due to errors in the plane fitting on the palm region because of the presence of several gestures having the fingers tightly folded on the whole palm region thus disrupting the palm flatness.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.6	0.13	0.06	0	0.04	0.1	0.01	0.06	0	0
G2	0.26	0.43	0.12	0.02	0.03	0.09	0.01	0.01	0	0.03
G3	0.1	0.28	0.33	0.04	0.01	0.08	0	0.09	0	0.07
G4	0	0.01	0.05	0.81	0.06	0.05	0.01	0	0.01	0
G5	0.03	0.06	0.05	0.08	0.61	0.12	0.03	0	0.01	0.01
G6	0.07	0.01	0.02	0.01	0.04	0.83	0	0.01	0.01	0
G7	0	0	0	0.01	0.11	0.01	0.65	0	0.16	0.06
G8	0.14	0.09	0.06	0	0.02	0.01	0.02	0.56	0.01	0.09
G9	0.02	0	0.03	0.02	0.08	0.04	0.05	0	0.67	0.09
G10	0	0.01	0.05	0.03	0.04	0.02	0.01	0.04	0.14	0.66

(a) MICROSOFT dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.56	0.04	0.06	0.01	0.01	0.04	0.01	0.01	0.12	0.01	0.01	0.12
G2	0.04	0.43	0.01	0.1	0	0.02	0.08	0.05	0.01	0	0.01	0.26
G3	0.2	0.03	0.13	0.02	0.13	0.02	0.1	0.01	0.24	0.05	0.01	0.06
G4	0.04	0.21	0	0.57	0.04	0.01	0.04	0.01	0	0.01	0	0.06
G5	0.07	0.06	0.11	0.03	0.3	0.03	0.15	0.04	0.13	0	0.04	0.05
G6	0.1	0.01	0.02	0.01	0.05	0.59	0.03	0.01	0.03	0.09	0.01	0.05
G7	0.08	0.07	0.15	0.01	0.11	0.02	0.19	0.12	0.11	0	0.06	0.06
G8	0.08	0.02	0.04	0.06	0.09	0.01	0.17	0.34	0.06	0.02	0.02	0.08
G9	0.14	0.04	0.12	0.01	0.08	0.01	0.06	0	0.46	0	0.01	0.08
G10	0.06	0.01	0.02	0.04	0.03	0.19	0.04	0.03	0.01	0.44	0.06	0.08
G11	0.05	0.11	0.01	0	0.09	0	0.04	0.01	0.03	0.01	0.66	0.01
G12	0.05	0.12	0	0	0.01	0.01	0.01	0	0.01	0	0	0.79

(b) LTTM dataset

Figure 7.6: Palm morphology features

Analogously to the hand contour distances from the palm plane, this descriptor is affected as well by the different quality in the plane fitting and the fact that in most gestures several fingers lay close to the palm plane not carrying any relevant information.

Hand contour similarity features ($\mathbf{F^z}$) is another descriptor reporting a ratherly high accuracy (beyond 87% on MICROSOFT dataset), comparable to the one of the hand contour distances from the palm center feature set. This is another expected result, since the correlation values the descriptor is made of are at the basis of the distance descriptor of Section 5.1.1 but contain less information. As the selected similarity measure is, in fact, the zero-mean cross-correlation, the distance plot amplitude information is not relevant for the hand contour alignment with the gesture templates. For this reason, $\mathbf{F^z}$ is not able to discriminate certain gestures characterized by the same raised finger configuration, e.g. Fig. 7.7(b) shows that G9 is misclassified as G3 most of the times in LTTM dataset where the two gestures only differ for the opening status of the finger. Fig. 7.7(a) analogously shows that, because of distance information loss, G1 of MICROSOFT dataset is often mistaken as other gestures, an ambiguity not found in other descriptors. However, thanks to the small descriptor size and its fast computation, this feature set can be considered for applications where the execution time and the memory footprint of the descriptors are critical.

The hand contour similarity features exploiting the sum of squared distances in place of ZNCC ($\mathbf{F^s}$) have very poor performance on all the three datasets and the reason is the high amount of noise on the computed distance plots, amplified by the square of the plot differences the descriptor is based on. Recall, in fact, that SSD is defined as the point-wise sum of the differences between two vectors of measures, and ideally tends to 0 when the vectors are similar. Nonetheless this descriptor, as will be further shown, is able to improve the overall gesture recognition accuracy when combined with proper feature sets.

Hand contour curvatures ($\mathbf{F^c}$) is one of the most performant descriptors that alone is able to discriminate most of the gestures in the selected datasets, thanks to its capability of detecting the concavities and convexities characterizing the hand contour. Moreover, since the curvatures only rely on the hand depth mask and not on the hand orientation, the palm plane fitting or on detected palm centroid position, this descriptor allows extremely high recognition accuracies even in challenging situations where the estimation of the hand orientation is not always accurate or even possible. Fig. 7.9 compares the performance of this descriptor on all the three datasets.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.6	0.15	0.1	0.02	0	0	0.06	0.06	0.01	0
G2	0.05	0.92	0	0	0	0	0.02	0	0.01	0
G3	0.07	0.01	0.79	0.12	0.01	0	0	0	0	0
G4	0.01	0	0.03	0.93	0.02	0.01	0	0	0	0
G5	0	0	0.03	0.06	0.9	0.01	0	0	0	0
G6	0	0	0	0	0.02	0.98	0	0	0	0
G7	0.05	0	0	0	0	0	0.94	0	0.01	0
G8	0.08	0	0	0.01	0	0	0.03	0.88	0	0
G9	0.04	0.05	0	0	0	0	0.01	0.04	0.85	0.01
G10	0.01	0	0	0	0.01	0.02	0	0.01	0	0.95

(a) MICROSOFT dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.37	0.09	0.06	0.03	0.13	0.07	0.06	0.08	0.05	0.04	0	0.02
G2	0.13	0.63	0	0.01	0	0	0.01	0.2	0.01	0	0.01	0
G3	0.04	0.01	0.55	0.01	0.14	0.06	0.01	0	0.19	0	0	0
G4	0.04	0.06	0	0.56	0.02	0	0.04	0.19	0.03	0.01	0	0.05
G5	0.2	0.01	0.15	0.01	0.34	0.02	0.01	0.02	0.19	0.04	0	0
G6	0.06	0	0.03	0.01	0.05	0.74	0.01	0	0.06	0.03	0.01	0
G7	0.09	0.05	0	0.05	0.02	0	0.56	0.21	0	0	0	0.01
G8	0.03	0.16	0	0.11	0	0	0.21	0.47	0.01	0	0	0.01
G9	0.18	0.06	0.33	0	0.17	0.05	0	0.03	0.19	0	0	0
G10	0.05	0	0	0	0	0.02	0	0	0	0.91	0.02	0
G11	0	0.01	0	0	0	0	0	0	0	0.01	0.97	0.01
G12	0.01	0	0	0.01	0	0	0	0.01	0	0.01	0.01	0.95

(b) LTTM dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.51	0.13	0.02	0.01	0.06	0.09	0.02	0.06	0.06	0.05
G2	0.05	0.61	0.14	0.04	0.01	0	0.01	0.03	0.04	0.07
G3	0.03	0.14	0.56	0.03	0	0.03	0.09	0.08	0.03	0.02
G4	0	0.04	0.06	0.66	0	0	0.06	0.16	0	0.01
G5	0.06	0.08	0.03	0	0.64	0.06	0	0.01	0.06	0.06
G6	0.04	0.08	0.01	0	0.06	0.73	0	0	0.05	0.03
G7	0	0.03	0.03	0.01	0	0	0.77	0.11	0.04	0.01
G8	0	0	0.08	0.1	0	0.03	0.16	0.51	0.06	0.06
G9	0.03	0.07	0	0	0.06	0	0.02	0.09	0.37	0.36
G10	0.06	0.1	0.05	0.01	0.11	0.02	0.01	0.03	0.26	0.35

(c) LEAPNECT dataset

Figure 7.7: Hand contour similarity (with ZNCC)

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.98	0	0	0	0	0	0	0	0.02	0
G2	0	0.25	0	0.34	0.12	0.05	0.04	0.2	0	0
G3	0	0.01	0.64	0.06	0.12	0.17	0	0	0	0
G4	0.01	0.12	0	0.47	0.18	0	0.03	0.1	0.09	0
G5	0	0.05	0.54	0.13	0.14	0.11	0	0.02	0	0.01
G6	0	0.06	0.42	0.01	0.16	0.31	0	0.03	0	0.01
G7	0	0.05	0	0.12	0.09	0	0.39	0.06	0.29	0
G8	0	0.12	0.01	0.18	0.04	0.06	0.03	0.42	0.03	0.11
G9	0.03	0	0	0.06	0	0	0.32	0.04	0.54	0.01
G10	0	0	0	0	0.02	0.02	0	0.13	0.02	0.81

(a) MICROSOFT dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.71	0.07	0.01	0	0	0.13	0	0	0	0.01	0.06	0
G2	0.11	0.26	0.04	0.19	0.01	0.19	0.09	0	0	0.07	0.03	0.01
G3	0.01	0.06	0.19	0.24	0.16	0.04	0.08	0.06	0.01	0.01	0.11	0.02
G4	0	0.17	0.11	0.41	0.02	0	0.08	0.04	0.06	0.04	0.02	0.04
G5	0	0.04	0.06	0.16	0.31	0.01	0.09	0.11	0.16	0.01	0.03	0.04
G6	0.26	0.16	0.11	0.01	0.01	0.25	0.01	0	0	0.06	0.11	0
G7	0.01	0.05	0.05	0.1	0	0.04	0.34	0.05	0.01	0	0.09	0.26
G8	0	0	0.02	0.06	0.09	0	0.06	0.34	0.41	0.01	0	0
G9	0	0.01	0.03	0.11	0.03	0	0.08	0.31	0.38	0	0.02	0.03
G10	0.09	0.26	0.14	0.14	0.03	0.06	0.06	0.04	0	0.09	0.1	0
G11	0.04	0.01	0.09	0	0.04	0.09	0.04	0	0	0	0.62	0.06
G12	0	0.01	0.02	0.04	0	0.01	0.18	0.01	0	0.01	0.08	0.64

(b) LTTM dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.42	0.26	0.06	0.01	0.07	0.13	0	0	0	0.05
G2	0.21	0.25	0	0.01	0.18	0.16	0	0.02	0	0.17
G3	0.01	0.01	0.16	0.16	0.12	0.01	0.12	0.08	0.11	0.21
G4	0	0.06	0.08	0.33	0.09	0.01	0.07	0.09	0.06	0.2
G5	0.01	0.01	0.13	0.14	0.17	0.09	0.15	0.09	0.05	0.16
G6	0.15	0.14	0.01	0.01	0.17	0.27	0.06	0.01	0	0.17
G7	0	0	0.09	0.14	0.1	0.11	0.34	0	0.13	0.09
G8	0	0.04	0.08	0.2	0.08	0.01	0.07	0.29	0.21	0.03
G9	0	0	0.09	0.16	0.11	0.01	0.15	0.14	0.29	0.06
G10	0.02	0.07	0.09	0.09	0.05	0.1	0.06	0.03	0.05	0.44

(c) LEAPNECT dataset

Figure 7.8: Hand contour similarity (with SSD)

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	1	0	0	0	0	0	0	0	0	0
G2	0	0.98	0	0	0	0	0	0	0.02	0
G3	0	0	0.87	0.01	0	0	0.01	0.1	0	0.01
G4	0	0	0	0.92	0.07	0	0	0	0	0.01
G5	0	0	0	0.08	0.77	0.15	0	0	0	0
G6	0	0	0	0	0.1	0.9	0	0	0	0
G7	0	0	0.01	0	0	0	0.95	0.03	0.01	0
G8	0	0	0.13	0	0	0	0.03	0.84	0	0
G9	0	0.06	0	0	0	0	0.01	0.01	0.92	0
G10	0	0	0	0.04	0	0	0	0	0	0.96

(a) MICROSOFT dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.95	0.04	0	0	0	0	0	0	0.01	0	0	0
G2	0.07	0.89	0	0.01	0.01	0.01	0	0	0.01	0	0	0
G3	0	0	0.85	0	0.01	0.04	0	0	0.11	0	0	0
G4	0	0.04	0	0.86	0	0	0.01	0.1	0	0	0	0
G5	0	0.01	0	0	0.97	0	0.01	0	0.01	0.01	0	0
G6	0	0.01	0.01	0	0.01	0.81	0	0.01	0.13	0.01	0	0
G7	0	0	0	0.03	0	0	0.9	0.03	0	0.02	0.02	0
G8	0	0	0	0.04	0	0.01	0.03	0.89	0	0	0.01	0.02
G9	0.02	0	0.09	0.01	0	0.11	0	0	0.76	0.01	0	0
G10	0	0	0	0	0	0.01	0.02	0	0.01	0.96	0	0
G11	0	0	0.01	0.03	0	0.01	0.01	0	0	0.01	0.94	0
G12	0	0.01	0	0.02	0	0	0	0.01	0	0	0	0.96

(b) LTTM dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.99	0	0.01	0	0	0	0	0	0	0
G2	0	0.89	0.06	0.01	0.01	0.01	0.01	0	0	0
G3	0.01	0.11	0.8	0.01	0.01	0.01	0.05	0.01	0	0
G4	0	0.03	0	0.91	0	0	0.04	0.02	0	0
G5	0	0	0.01	0.04	0.89	0.03	0.02	0	0	0.01
G6	0	0	0.02	0	0.04	0.74	0.01	0	0.01	0.19
G7	0	0.02	0.06	0.04	0.02	0	0.8	0.05	0	0
G8	0	0	0	0.01	0.01	0.01	0.04	0.85	0.01	0.08
G9	0	0	0	0	0	0.02	0.01	0.01	0.96	0
G10	0	0	0	0	0	0.18	0.01	0.01	0	0.81

(c) LEAPNECT dataset

Figure 7.9: Hand contour curvatures

Convex hull area ($\mathbf{F^{ca}}$) and perimeter ratios ($\mathbf{F^{cp}}$) are the least performing features because they are only made by a single scalar number that is not sufficient to separate all the gesture classes. The separation is also challenging for the measurement noise corrupting the detected hand contour.

The convex hull connected component area ratios ($\mathbf{F^{cc}}$) are, instead, one of the most performant descriptors (with a recognition accuracy up to the 72%) in spite of their small number. The ratherly high reached recognition accuracy is a proof of the relevant amount of information encoded in the hand convex hull, as already stated in Section 5.1.6. Again, its small size and simple computation makes this descriptor interesting when a trade-off between performance and accuracy is required, especially when the hand contour similarity features are not easy to extract.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.96	0.04	0	0	0	0	0	0	0	0	0	0
G2	0.19	0.69	0	0	0.01	0.1	0	0.01	0.01	0	0	0
G3	0	0.01	0.52	0	0.16	0.07	0.04	0	0.1	0.05	0.04	0
G4	0	0.01	0	0.68	0	0	0.03	0.18	0.03	0.03	0	0.04
G5	0	0.02	0.09	0	0.78	0.03	0	0	0.04	0.05	0	0
G6	0	0.05	0.04	0	0.01	0.79	0	0	0.06	0.04	0.01	0
G7	0	0	0.04	0.05	0	0	0.74	0.07	0.02	0.04	0.03	0.01
G8	0.01	0	0	0.17	0	0.01	0.05	0.7	0.01	0.03	0	0.01
G9	0.01	0.07	0.09	0.04	0.15	0.11	0.01	0.01	0.49	0.01	0	0
G10	0	0.01	0.08	0.04	0.09	0.06	0.07	0.09	0.04	0.51	0	0.01
G11	0	0	0.05	0	0	0	0.02	0	0	0.06	0.86	0
G12	0	0.01	0	0.04	0	0	0.01	0	0.01	0	0	0.93

(a) LTTM dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.99	0	0.01	0	0	0.01	0	0	0	0
G2	0.03	0.46	0.26	0.01	0.09	0.03	0.06	0.02	0	0.04
G3	0.01	0.21	0.68	0.01	0.03	0.02	0.04	0	0	0
G4	0	0.03	0	0.67	0.19	0	0.02	0.08	0	0.01
G5	0	0.08	0.01	0.2	0.59	0.06	0	0	0.01	0.05
G6	0.01	0.06	0.01	0.01	0.01	0.8	0.01	0.01	0.03	0.04
G7	0	0.09	0.12	0.01	0.01	0.09	0.64	0	0.01	0.03
G8	0	0.04	0	0.09	0.02	0.01	0	0.69	0	0.15
G9	0	0	0.01	0	0.01	0.04	0	0.01	0.94	0
G10	0	0.09	0.01	0.01	0.05	0.01	0.04	0.15	0	0.64

(b) LEAPNECT dataset

Figure 7.10: Convex hull connected components area ratios

Although certain depth feature descriptors have, when used alone, the power of reliably recognize most of the performed gestures (e.g., hand contour distances from the palm center or hand contour curvatures), the proper combination of two or more compatible feature sets can boost the overall recognition accuracy even though one of the descriptors has poor performance. Two or more feature sets are compatible when they are poorly correlated and thus they do not carry redundant information when combined. Moreover, compatible descriptors are able to compensate the counterpart failures in most cases, namely they can correctly classify certain feature vectors misclassified by other descriptors and viceversa.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.82	0.04	0	0.02	0	0	0.05	0.05	0.02	0
G2	0	0.98	0.01	0	0	0	0	0	0.01	0
G3	0	0.03	0.87	0.02	0.08	0	0	0	0	0
G4	0	0	0.02	0.96	0	0	0	0.02	0	0
G5	0	0	0.02	0.05	0.88	0.05	0	0	0	0
G6	0	0	0	0	0.02	0.98	0	0	0	0
G7	0.05	0	0	0	0	0	0.94	0	0.01	0
G8	0.02	0	0	0	0	0	0.03	0.95	0	0
G9	0.04	0.05	0	0	0	0	0.01	0.03	0.87	0
G10	0	0	0	0	0.01	0.02	0	0.01	0	0.96

(a) MICROSOFT dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.66	0.14	0	0	0.02	0.11	0.01	0	0	0.04	0	0.02
G2	0.09	0.86	0	0.02	0.01	0	0.01	0	0	0	0	0
G3	0.01	0	0.65	0	0.26	0.01	0.01	0	0.06	0	0	0
G4	0	0.09	0.03	0.76	0.02	0	0.04	0.05	0	0	0.01	0.01
G5	0	0.01	0.25	0.03	0.56	0.01	0	0.04	0.08	0.01	0.01	0
G6	0.14	0.01	0.04	0	0.01	0.79	0.01	0	0	0.02	0	0
G7	0.04	0	0	0.04	0.01	0	0.87	0.01	0.01	0	0	0.02
G8	0	0.01	0	0.07	0.01	0	0.1	0.74	0.05	0.01	0	0
G9	0	0	0.08	0.01	0.12	0.01	0	0.11	0.67	0	0	0
G10	0.01	0	0	0	0.01	0.06	0	0	0	0.87	0.04	0
G11	0	0	0	0	0	0	0	0	0	0.01	0.98	0.01
G12	0	0	0	0.01	0	0	0	0	0	0	0	0.99

(b) LTTM dataset

Figure 7.11: Combination of hand contour similarity features

For example, the hand contour distances from the palm center and the curvature features are highly compatible and augment the recognition accuracies when jointly used, but a dramatically higher improvement is given by the combination of the two similarity feature sets, leading to an overall recognition accuracy increment of the 6% for MICROSOFT dataset and almost the 20% for LTTM.

Note how in Fig.7.11(a), for example, the SSD metric removed most of the ambiguities in discriminating G1 from G2 in MICROSOFT dataset, and the beneficial effect is even more evident in Fig. 7.11(b) for LTTM dataset.

As already stated in Section 5.1.3, the improvement is due to the fact the ZNCC is not able to capture the hand contour amplitude difference but is robust to noise, while the SSD is highly sensitive to noise but able to discriminate similar gestures with a different distance plot amplitude.

In other cases, instead, the overall improvement in recognition accuracy is neglectable, like the combination of the curvature and the contour similarity features in the LEAPNECT dataset that only led to an accuracy increment around the 2% since the two descriptors are weakly decoupled. This effect is more evident in the last rows of Table 7.1, where adding more feature sets did not lead to any sensible improvement or, in limited cases, the elevated number of features only introduced clutter with detrimental effects on the overall recognition accuracy.

Convex hull features are another set of weakly compatible descriptors that when combined slightly improved the overall performance on LTTM dataset, while on LEAPNECT dataset the increment in accuracy is higher.

Table 7.2 reports the accuracies of the Leap Motion feature sets extracted from the LEAPNECT dataset.

Feature set	Estimated accuracy (%)
$\mathbf{F_L^d}$	76.1
$\mathbf{F_L^e}$	73.1
$\mathbf{F_L^\theta}$	74.2
$\mathbf{F_L^p}$	81.5
$\mathbf{F_L^d} + \mathbf{F_L^e}$	80.2
$\mathbf{F_L^d} + \mathbf{F_L^\theta}$	73.8
$\mathbf{F_L^e} + \mathbf{F_L^\theta}$	77.3
$\mathbf{F_L^d} + \mathbf{F_L^e} + \mathbf{F_L^\theta}$	80.9

Table 7.2: Comparison of the Leap Motion features accuracies on LEAPNECT

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
G1	0.94	0.06	0	0	0	0	0	0	0	0	0	0
G2	0.23	0.66	0	0	0	0.08	0	0	0.01	0.02	0	0
G3	0	0	0.54	0	0.16	0.11	0	0	0.11	0.08	0	0
G4	0	0.02	0	0.63	0	0	0.04	0.29	0.01	0	0	0.01
G5	0	0.01	0.07	0	0.73	0.06	0	0	0.07	0.06	0	0
G6	0	0.06	0.05	0.01	0.01	0.69	0	0	0.11	0.08	0.01	0
G7	0	0	0	0.04	0	0	0.84	0.02	0	0.03	0.06	0.01
G8	0	0	0	0.2	0	0	0.04	0.72	0.01	0.01	0.01	0.01
G9	0.01	0.08	0.07	0.01	0.16	0.13	0.01	0	0.5	0.03	0	0
G10	0	0.03	0.06	0.01	0.06	0.06	0.06	0	0.04	0.67	0	0
G11	0	0	0	0	0	0	0.03	0.01	0	0.04	0.92	0
G12	0	0.01	0	0.02	0	0	0	0.01	0	0	0	0.95

(a) LTTM dataset

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.98	0	0.01	0	0	0.01	0	0	0	0
G2	0.03	0.56	0.25	0.06	0.01	0.01	0.08	0	0	0.01
G3	0.02	0.16	0.76	0.01	0.01	0.01	0.04	0.01	0	0
G4	0	0.06	0	0.76	0.11	0	0.01	0.05	0	0
G5	0	0.02	0	0.12	0.81	0	0.01	0.01	0	0.01
G6	0.02	0.01	0	0	0.04	0.85	0.03	0	0.01	0.04
G7	0	0.09	0.07	0.01	0.02	0.01	0.79	0.01	0	0.01
G8	0	0	0	0.04	0.06	0	0.01	0.79	0	0.1
G9	0	0	0	0	0	0.03	0	0.02	0.95	0
G10	0	0	0	0	0.03	0.04	0.01	0.04	0	0.89

(b) LEAPNECT dataset

Figure 7.12: Combination of all the convex hull features

Features coming from the Leap Motion data report on the overall high accuracies also for the analogue descriptors on depth data poorly performing, like the distances of the fingertips from the palm plane.

A noteworthy result in Table 7.2 are the similar accuracies reported for the distances of the fingertips from the hand center ($\mathbf{F}_{\mathbf{L}}^{\mathbf{d}}$), the palm plane ($\mathbf{F}_{\mathbf{L}}^{\mathbf{e}}$) and their orientations respect to the main axis ($\mathbf{F}_{\mathbf{L}}^{\theta}$). In particular, the fingertip distances from the palm plane that, when extracted from depth data returned ratherly low accuracies, in this case can be used in place of the distances from the hand center for a reliable gesture recognition.

A possible reason of this favorable behavior may be due to the same construction of the first three descriptors in Table 7.2 and to the fact they are probably

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.89	0.02	0.07	0.01	0	0	0	0	0	0.01
G2	0.3	0.49	0.21	0.01	0	0	0	0	0	0
G3	0.17	0.27	0.48	0.06	0	0	0.02	0	0	0
G4	0.04	0.01	0.03	0.66	0.03	0	0.1	0.04	0	0.09
G5	0.05	0.07	0.02	0	0.82	0	0	0.01	0	0.02
G6	0.01	0.01	0.03	0	0.01	0.91	0	0	0.01	0.03
G7	0.01	0.01	0.02	0.09	0.04	0.03	0.79	0.01	0	0
G8	0	0	0.01	0.06	0.05	0	0.03	0.81	0	0.04
G9	0	0	0.01	0	0	0	0	0.01	0.96	0.02
G10	0.01	0.01	0.01	0.06	0.01	0.07	0	0.01	0.03	0.8

Figure 7.13: Fingertip distances from the hand center (Leap Motion)

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.88	0.01	0.09	0.01	0	0	0	0	0	0.01
G2	0.31	0.52	0.15	0.02	0	0	0	0	0	0
G3	0.17	0.01	0.69	0.06	0.04	0.02	0.01	0	0	0
G4	0.01	0.01	0.04	0.77	0.06	0	0.06	0.02	0	0.02
G5	0.06	0.09	0.03	0.03	0.69	0	0	0.04	0	0.06
G6	0.01	0	0.03	0	0.02	0.76	0.02	0	0.01	0.14
G7	0.01	0.01	0.03	0.07	0.04	0.02	0.62	0.15	0	0.05
G8	0	0	0	0.01	0.06	0	0.06	0.81	0.04	0.02
G9	0.01	0	0	0	0	0.01	0.03	0.04	0.9	0.01
G10	0.01	0.01	0.01	0	0.1	0.12	0.03	0.05	0.01	0.66

Figure 7.14: Fingertip distances from the palm plane (Leap Motion)

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.86	0.06	0.06	0.03	0	0	0	0	0	0
G2	0.3	0.59	0.09	0.01	0.01	0	0	0	0	0
G3	0.11	0.15	0.6	0.09	0.05	0	0	0	0	0.01
G4	0.01	0	0.02	0.84	0.01	0	0.09	0.01	0	0.01
G5	0.05	0.06	0.02	0.01	0.76	0	0.01	0.04	0	0.05
G6	0.01	0	0.04	0.01	0.02	0.82	0	0	0.01	0.09
G7	0.01	0	0.03	0.17	0.01	0.01	0.56	0.16	0	0.06
G8	0	0	0	0.03	0.04	0	0.08	0.8	0.03	0.02
G9	0.01	0	0	0.01	0	0.04	0	0.04	0.9	0.01
G10	0	0	0.02	0.04	0.07	0.13	0.02	0.02	0	0.69

Figure 7.15: Fingertip orientations (Leap Motion)

more characterized by the fingertip occupation of the regions in Fig. 5.20 than from the measured feature values. A proof of this statement can be found in Fig. 7.3, where the reported gestures are clearly characterized by the number and the configuration of the raised fingers in most cases.

The 3D positions of the fingertips ($\mathbf{F_L^p}$) are the most performing descriptor extracted from the Leap Motion data (accuracy of 81.5%), although the difference in accuracy with the other descriptors is limited respect to the differences measured in the feature sets extracted from the depth data. This descriptor, alone, is able to reliably recognize most of the performed gestures, although it may fail in the discrimination of gestures like G2 and G3 (Fig. 7.16).

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.89	0.02	0.06	0.02	0	0	0	0	0	0
G2	0.3	0.56	0.14	0	0	0	0	0	0	0
G3	0.14	0.09	0.7	0.04	0	0	0.02	0	0	0
G4	0.03	0	0	0.9	0	0	0.05	0	0.01	0.01
G5	0.05	0.05	0.03	0.02	0.76	0.01	0.02	0.02	0	0.04
G6	0.01	0	0.03	0	0.03	0.84	0.01	0.01	0	0.07
G7	0.01	0	0.04	0.08	0	0	0.81	0.03	0.01	0.02
G8	0	0	0	0.04	0.03	0	0.03	0.83	0	0.08
G9	0	0.01	0	0.01	0	0	0	0.01	0.97	0
G10	0	0	0.01	0	0.04	0.01	0.05	0.01	0	0.89

Figure 7.16: Fingertip positions (Leap Motion)

It is worth noting that the fingertip positions are, again, a feature set containing all the information carried singularly by the remaining descriptors, and this is proved by the fact that the joint usage of ($\mathbf{F_L^d}$), ($\mathbf{F_L^e}$) and ($\mathbf{F_L^\theta}$) reports almost the same recognition accuracy obtained with ($\mathbf{F_L^p}$) alone.

Finally, the combination of a few most performing feature sets from the two devices, that is $\mathbf{F_2^l}$, $\mathbf{F^c}$ and $\mathbf{F_L^p}$, led to a recognition accuracy of 96.5%, proving the Leap Motion features allow to improve the overall performance with the minimal effort. Note how the improvement in this case is very limited both because the employed distance descriptor alone reports an accuracy higher than the 94% and the other feature sets do not add any useful new information to aid the gesture recognition.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	0.89	0.02	0.07	0.01	0	0	0	0	0	0.01
G2	0.3	0.51	0.19	0.01	0	0	0	0	0	0
G3	0.16	0.06	0.68	0.04	0	0.01	0.02	0	0	0.02
G4	0.04	0.01	0.04	0.74	0.01	0	0.04	0.04	0	0.09
G5	0.05	0.06	0.02	0.02	0.8	0	0	0.02	0	0.02
G6	0.01	0	0.03	0	0.02	0.91	0	0	0.01	0.03
G7	0.01	0	0.05	0.05	0.03	0.01	0.8	0.04	0	0.01
G8	0	0	0	0.01	0.05	0.01	0.03	0.84	0	0.06
G9	0	0	0.01	0	0	0	0	0.01	0.97	0.01
G10	0	0	0.01	0.03	0.02	0.02	0	0.01	0.03	0.88

Figure 7.17: Combination of fingertip distances from the hand center and from the palm plane (Leap Motion)

7.2 Ensembles of classifiers performance

Section 7.1 analyzed the performance of a single SVM classifier trained on the extracted features from the depth and the Leap Motion data collected in three selected datasets. The classifier parameters were optimized with a variation of grid-search method tailored to the gesture recognition purpose and the combination of various feature sets was performed by simply juxtaposing the feature vectors of each set.

In particular, Tables 7.1 and 7.2 showed that, because of the feature correlation, the juxtaposition of proper uncorrelated feature sets boosted sensibly the overall recognition accuracy, while in other cases the improvement was neglectable. In the worst situations, the dimensionality curse caused by the blind combination of very long feature vectors could even affect negatively the overall recognition accuracy introducing novel ambiguities.

For this reason, the tests performed in current section aim at evaluating the overall improvement in accuracy that can be obtained by replacing the single SVM classifier with a proper ensemble of Section 6.3, designed to leverage the capabilities of each feature set considered singularly.

Differently from Section 7.1, which only considered geometrical features, this section also evaluates the performance of the textural descriptors analyzed in Section 5.4. In this case HOG ($\mathbf{F}^{\mathbf{hog}}$), LPQ ($\mathbf{F}^{\mathbf{lpq}}$) and LTP ($\mathbf{F}^{\mathbf{ltp}}$) feature extraction algorithms either ran on the L channel of the hand image expressed in Lch color space [66, 64] or the grayscale image obtained by considering the curvature values (after rearranging the curvature descriptor in a 2D matrix as in

Fig. 5.10) as pixel intensities. The texture descriptors were extracted from 50 random reshapings [66, 64] of the hand contour curvatures in a 2D matrix and the accuracy obtained by the sum-rule on the 50 single classifiers trained on each reshaped image. Using multiple reshapings allows to observe and encode different aspects of the curvature variations from a single curvature vector.

As already stated in Section 5.4, the HOG descriptor used separate blocks of 5×6 cells generating histograms of 9 bins 20° wide, accounting for gradient directions from 0 to 180°. LPQ descriptor was made, instead, by the juxtaposition of two single LPQ descriptors extracted within patches of 3×3 and 5×5 pixels respectively in order to capture different properties of the underlining local phases. LTP followed the same rationale using 8 and 16 neighborhood connectivity in each texture patch, exploiting the NPE variation in place of PCA.

A motivation that lead to the evaluation of ensembles of classifiers is due to the Q statistics [91] performed on the MICROSOFT and LTTM datasets using a RS of SVM ensemble. The results are reported in Table 7.3.

Feature set	$\mathbf{F^d}$	$\mathbf{F^c}$	$\mathbf{F^a}$	$\mathbf{F^e}$	$\mathbf{F^{lpq}}$
$\mathbf{F^c}$	-	-	0.32	0.27	0.37
$\mathbf{F^a}$	-	-	-	0.27	0.22
$\mathbf{F^e}$	-	-	-	-	0.32
$\mathbf{F^{lpq}}$	-	-	-	-	-

Table 7.3: Q statistics on selected feature sets

Since the collected values are rather low, according to the Q statistics the considered feature sets are poorly correlated thus proving their combination can lead to significant improvement on the overall recognition accuracy, especially by exploiting ensembles of classifiers.

Table 7.4 collects the accuracies of the tests performed on the curvature features using a RS of SVM ensemble. The first three rows are referred respectively to the curvature feature vectors of Section 5.1.4 and the HOG and LPQ descriptors extracted on its reshapings in grayscale images, while the remaining rows collect the results of the weighted combination of the curvature vectors with the best performing texture descriptor. Note how the weights modulate the impacts of the single descriptor votes in the final ensemble decision.

Table 7.4 shows, again, that the hand contour curvatures are an highly performing feature set that, alone, can reliably recognize almost all the gestures in the considered datasets, but when combined with other descriptors can improve

	Estimated accuracy (%)	
Feature set	MICROSOFT	LTTM
$\mathbf{F^c}$	92.4	82.7
$\mathbf{F^{hog}}$	94.7	84.6
$\mathbf{F^{lpq}}$	91	80.9
Curvature + $\mathbf{F^{hog}}$	94.5	86.2
Curvature + $2 \times \mathbf{F^{hog}}$	94.9	86.5
Curvature + $3 \times \mathbf{F^{hog}}$	94.8	86.0
Curvature + $4 \times \mathbf{F^{hog}}$	94.8	85.6

Table 7.4: Performance of the curvature feature with RS of SVM on two datasets

the overall recognition accuracy. In particular, the HOG descriptor in this case demonstrated to the more performing than LPQ and slightly more reliable than the curvature feature vector, since the assignation of an higher weight in the combination led to a modest improvement on the ensemble performance.

Table 7.5 compares, instead, the performance of different ensembles of classifiers operating on the same feature sets, where $\mathbf{F^{ct}} = \mathbf{F^c} + 2 \times \mathbf{F^{hog}}$ is a weighted combination of depth and texture descriptors and $\mathbf{F^{tex}} = 4 \times \mathbf{F^{hog}} + \mathbf{F^{lpt}} + \mathbf{F^{ltp}}$ a pure weighted combination of texture descriptors only. Moreover, HET is an heterogeneous ensemble of RS of SVM and RS of ROTBOOST, and HET2 and heterogeneous ensemble where $\mathbf{F^c}$ is trained separately with a RS of SVM and the texture descriptors by aggregating with the sum rule the results of a SVM classifier trained on each reshaping.

Table 7.5, compared to Table 7.1, shows that a proper weighted ensemble trained on the same feature sets outperforms the single optimized classifier trained on long feature vectors while preventing the dimensionality curse.

		Estimated accuracy (%)	
Classification approach	Feature set	MICROSOFT	LTTM
RS of SVM	$\mathbf{F^d}$	86.9	57.2
	$\mathbf{F^e}$	60.5	46.2
	$\mathbf{F^c}$	92.4	84
	$\mathbf{F^a}$	60.9	45.3
	$\mathbf{F^{ct}}$	94.9	86.5
	$\mathbf{F^{tex}}$	95.5	88.1
	$\mathbf{F^d} + \mathbf{F^c}$	96.7	85.3
	$\mathbf{F^d} + 2 \times \mathbf{F^c}$	97.5	86.8
RS of ROTBOOST	$\mathbf{F^d}$	88.8	60.5
	$\mathbf{F^e}$	61.1	48.7
	$\mathbf{F^c}$	93.9	84.9
	$\mathbf{F^a}$	64.0	48.2
	$\mathbf{F^{tex}}$	94.7	88.0
	$\mathbf{F^d} + \mathbf{F^c}$	97.4	86.6
	$\mathbf{F^d} + 2 \times \mathbf{F^c}$	97.4	87.5
HET	$\mathbf{F^d}$	89	60.1
	$\mathbf{F^e}$	61.5	49.2
	$\mathbf{F^c}$	94.6	86.2
	$\mathbf{F^a}$	63.6	48
	$\mathbf{F^{tex}}$	95.3	89.3
	$\mathbf{F^d} + \mathbf{F^c}$	97.5	87.2
	$\mathbf{F^d} + 2 \times \mathbf{F^c}$	97.9	88.7
HET2	$\mathbf{F^{lpq}}$	91	80.9
	$\mathbf{F^{lpq}} + \mathbf{F^c}$	93.5	84.3
	$\mathbf{F^{lpq}} + 2 \times \mathbf{F^c}$	93.4	85
	$\mathbf{F^{lpq}} + 3 \times \mathbf{F^c}$	93.5	85.1
	$\mathbf{F^{lpq}} + 4 \times \mathbf{F^c}$	93.3	84.9

Table 7.5: Performance of different ensembles on the same features

7.3 Feature selection performance

This section compares the performance of the feature selection methods of Section 6.4 on the LEAPNECT dataset, with the aim of disclosing the minimal subset of features in $\mathbf{F^d} + \mathbf{F^c} + \mathbf{F^p_L}$ allowing to maintain the original level of accuracy (around 96.5%). The rationale consists in dramatically reducing the gesture recognition algorithm computational complexity while preserving its capability in discriminating the different gestures.

Fig. 7.18 shows several curves each one reporting, for a given classification algorithm, the average accuracy of the estimated gesture recognition model over the selected subsets of most relevant features. In order to avoid overfitting due to the low cardinality of the employed dataset, the feature selection has been performed following the rationale of Section 6.1. Namely, the tests performed on different subsets of features where repeated $M = 14$ times using one person's data as the validation set and the remaining data as training set, and by averaging the different models accuracies. The results for subsets of size 435, 128 and 16 features are reported in Table 7.6.

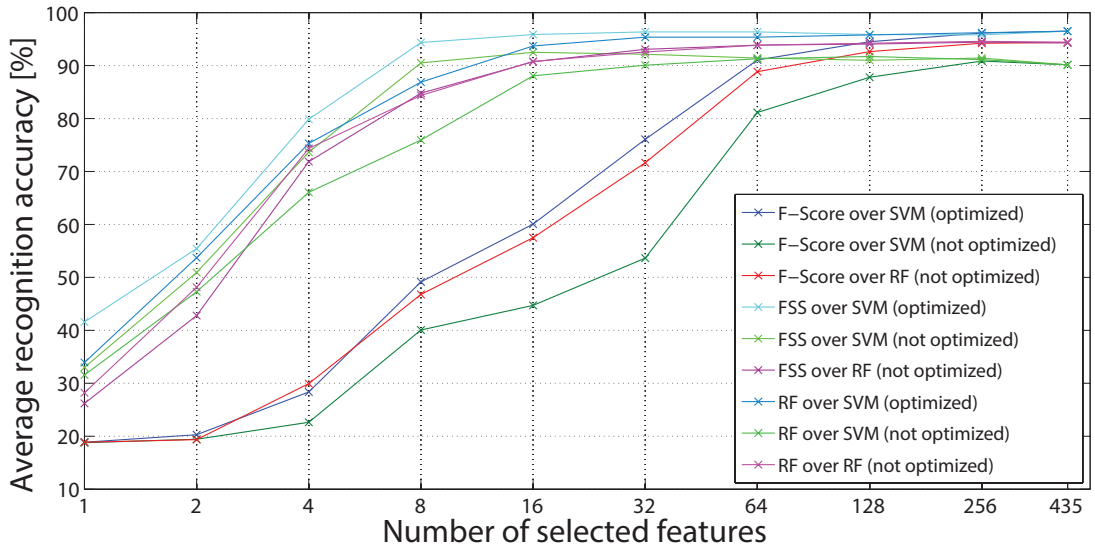


Figure 7.18: Performance of different feature selection algorithms on LEAPNECT

Fig. 7.18 shows that, as expected, F-score is the least performing metric, and the reason lies in the definition of F-score (Eq. 6.20). This measure is, in fact, unable to capture the possible mutual correlations between different features, as the assigned score to a given feature only depends on its value distribution within the considered training set. Furthermore, not only the average model accuracy increases along with the size of the selected features subset, compatibly with the

	SVM acc. (%)			RF acc. (%)		
Feature selection strategy	435	128	16	435	128	16
F-score	96.5	94.5	60.1	94.7	92.6	57.5
Sequential feature selection	96.5	95.9	95.8	94.7	94.1	90.7
Random forest	96.5	95.8	93.7	94.7	94.2	90.8

Table 7.6: Performance of different feature selection methods on LEAPNECT

previsions, but it is also evident that the grid-search is effective in this case as the optimized SVM classifier highly outperforms its unoptimized version. Compared to the other techniques, however, F-score does not lead to a sensible performance improvement, since the related curves show that the gesture recognition algorithm needs at least from $^1/_3$ to $^1/_2$ of the original features in order to not lose an excessive amount of information required to effectively classify the performed gestures.

The chart shows also that a not optimized Random Forest has, in general, a similar performance to the one of the optimized version of SVM. Random Forests for automatic gesture recognition are, then, a valid alternative to Support Vector Machines as they are faster to train (w.r.t. to the grid-search optimization) and less affected by the overfitting problem.

The remaining metrics, namely the Forward Sequential Selection (FSS) and the scoring according to the Out-of-Bag-Error (RF) of Random Forests, are clearly superior to the measure based on the F-score. In particular, FSS with an appropriate classifier is able to reach almost the maximum accuracy for the model with only a nearly 2% loss respect to the original feature set. This is due to the fact that FSS, differently from the F-score, is able to detect the mutual correlations between features as it selects at each round not the best feature according to an independent measure on its distribution on the dataset, but the feature that works better along with the previously selected ones.

Finally, FSS and RF have similar performance starting from a limited size features subset, although the two methods select features in a different way. For this reason, despite of being defined on Random Forests, the RF metric does not force the usage of Random Forest as the final classifier for the selected subset of features.

In conclusion of this section, it is worth analyzing an example of the first 8 and 16 selected feature indexes by FSS for one of the 14 trained models. Recall that a

generic feature vector $\mathbf{x_i}$ of the LEAPNECT dataset is made by the juxtaposition of 15 features for the fingertip positions (5 fingertips defined by 3 coordinates in the 3D space), 180 features for the hand contour distances from the palm center corresponding to an hand contour point sampled at intervals of 2°, and 240 features for the curvatures. The feature intervals are, then, $[1, 15]$ for the fingertip positions, $[16, 195]$ for the distances 2D and $[196, 435]$ for the curvatures.

The first 8 most relevant features are $2, 7, 46, 63, 79, 110, 138, 152$ and correspond to the y coordinate of the thumb, the x coordinate of the middle, the distances from the palm center at $(46 - 15 - 1)2 = 60°$, 94°, 126°, 188°, 244° and 272° respect to the main hand direction in counterclockwise order starting from the wrist center. FSS has selected, then, a pair of coordinates of the thumb and the middle finger, and a few hand boundary points at almost regular intervals of 60°. Note how no curvature feature has been selected; this does not mean the curvatures are not "good" features, but only that the fingertip positions and the hand boundary distances were enough to reliably discriminate most of the gestures.

A comparison with the first 16 selected features (1,2,7,41,42,45,47,63,80,99,108, 126,130,138,150,270) is a further proof of what stated: note how FSS only selected the first 8 relevant features and a few other features in their neighborhood. Only the one with index 270, corresponding to the 74-th curvature value, that is the 74 mod 10 = 4-th bin (that is a curvature value from 0.3 to 0.4) of the 74/10 = 7-th mask size probably adds new relevant information.

7.4 Algorithmic performance

The current chapter concludes with the analysis of the execution time of the proposed automatic gesture recognition algorithm processing MICROSOFT dataset.

The current implementation in C++ has not been fully optimized and has been tested on a not too performing desktop PC with an Intel Q6600 processor and 4Gb of RAM. Table 7.7 reports the measured average execution times of the main step in the recognition pipeline of Fig. 1.8.

The initial hand detection phase took only almost 7 mS on depth data only, while the magnitude order would be much higher if the color information was taken into account. One of the most time consuming phases for the moment is the palm detection, which currently takes around 60 mS. The plane fitting and the hand local coordinate system computation are, instead, very performing since they take less than 5 mS on the overall. Palm removal and other minor tasks

Task	Average execution time [ms]
Hand detection	6.6
Initial centroid detection	37.4
Palm detection	21.15
3D backprojection (full hand)	1.2
Arm removal	2.2
Point projection on palm plane	0.01
3D palm plane fitting	2.1
Palm coordinate system computation	2.1
Coordinate system change	1
Palm removal	0.4
Distances from palm centroid extraction	0.3
Distance plot smoothing	0.01
Distances from palm plane extraction	1
Curvatures extraction (circular masks)	349.3
Curvatures extraction (integral images)	18
Distance plot alignment	0.01
SVM classification	1

Table 7.7: Comparison of the average execution times on MICROSOFT dataset

related to geometrical transforms take neglectable times as well. Moreover, the extraction of the hand contour distances from the palm center, and consequently also the ones from the palm plane and the contour similarities are very efficient and do not increment the overall execution time sensibly. Thanks to the integral image expedient, the hand contour curvature descriptor extraction only requires a modest effort (less than 20 mS). Finally, the SVM classification of the extracted feature vector only takes 1 mS and does need further improvements.

The average execution time for the mandatory tasks and the extraction of two main descriptors on depth data takes less than 100 mS with the current implementation, corresponding to a frame rate higher than $10fps$. It is worth noting that an higher framerate (above $15fps$) can be obtained by the joint usage of a range camera and a Leap Motion, since most of the mandatory tasks are executed in few mS thanks to the data provided by the Leap Motion.

Finally, although the current implementation, not optimized, is able to reach pseudo real-time performance, and will be able to reach full real-time performance with the aid of a further optimization.

Chapter 8

Conclusions

Automatic hand gesture recognition is a very challenging task that has been raising an high interest in academy and industry due to its applicability in several fields, ranging from gaming to health care. In particular, hand gestures make possible the realization of novel 3D interfaces that allow a more natural interaction with computers and machinery without the need of a direct contact.

This thesis proposed a robust automatic gesture recognition framework that, differently from earlier approaches based on the processing of images and videos only or on the use of auxiliary devices to help the hand detection in the framed scene, is entirely based on the processing of depth and color data coming from low-cost range cameras (or the Leap Motion) framing the bare hand. The gesture recognition problem was, then, solved with state-of-art computer vision techniques thus preserving the naturalness of the interaction. The adopted gesture recognition pipeline follows four main steps consisting in firstly detecting the hand in the framed scene and segmenting it in palm and fingers regions, then in extracting several feature sets describing geometrical or textural properties of the hand and finally in recognizing the performed gesture with machine learning techniques.

The dissertation showed how depth data alone allows to quickly and reliably detect the hand in the framed scene when the hand is supposed to be the nearest object to the acquisition setup, and how the joint usage of depth and color information or depth and the Leap Motion data allows, instead, a reliable detection when this assumption is no longer valid. In particular, when the Leap motion data is available, it can be exploited to quicken the hand detection based on depth data. Then, the thesis described accurately how the palm is detected in the acquired depth maps and the steps followed for the construction of a local coordinate system centered on the palm center, which is at the basis of several fea-

ture extraction algorithms. Both palm detection and palm direction estimation algorithms take into account noise in the acquired data, exploiting the RANSAC framework and other methods. Several feature sets containing a considerable amount of information on the performed gesture, quantifying geometrical and textural properties of the hand contour and the palm morphology, were then analyzed. In particular, the depth information was also used to adapt the descriptors to the different hand sizes in order to make them robust to the user's hand anatomy. Novel features that can be easily and efficiently extracted from the Leap Motion data were also considered. Moreover, it was also shown how the Leap Motion data can be jointly used with depth information to extract the same depth based features with a significantly lower computation effort. Most of the proposed descriptors are characterized by a strong compatibility that allows their combination to improve the overall gesture recognition accuracy since they are highly uncorrelated. Various machine learning techniques, ranging from the consolidated Support Vector Machines (SVM) and Random Forests (RF) to more advanced a powerful ensembles of classifiers, were then described. In particular, it was shown how a proper variation of the grid-search approach allowed to optimize the SVM kernel parameters while reducing overfitting on small datasets. The results collected from several tests on three datasets and validated by proper metrics proved that the proposed framework is able, although not fully optimized, to recognize the performed gestures with extremely high accuracy. Moreover, the reduced execution times of the main steps of the recognition pipeline implemented on a real system proved the algorithm is capable of running in real-time.

Future work will improve the feature sets presented in this thesis by optimizing the extraction algorithms parameters, and will introduce novel descriptors leveraging depth, color and other types of information coming from low-cost color and range cameras, and the Leap Motion. Another part of the future work will be devoted to the improvement of the framework performance, since the current recognition algorithm implementation is not optimized and, although is capable of running in real-time with a limited memory footprint, it does not fully exploit the underlining hardware. By reinterpreting parts of the code more efficiently and by parallelizing independent tasks like the different feature sets extraction, in fact, the overall execution time will be sensibly reduced. Finally, the future work will also expand the existing framework to the recognition of dynamic gestures, a mandatory task in natural interfaces based on gestures. The existing feature sets and the tracking of the estimated hand centroid are a good starting point for this purpose.

Bibliography

[1] C. Dal Mutto, P. Zanuttigh, and G. M. Cortelazzo, *Time-of-Flight Cameras and Microsoft Kinect*. SpringerBriefs, Springer, 2012.

[2] C. Dal Mutto, F. Dominio, P. Zanuttigh, and G. M. Cortelazzo, "Hand gesture recognition for 3d interfaces," 2011.

[3] J. P. Wachs, M. Kölsch, H. Stern, and Y. Edan, "Vision-based hand-gesture applications," *Commun. ACM*, vol. 54, pp. 60–71, Feb. 2011.

[4] P. Garg, N. Aggarwal, and S. Sofat, "Vision based hand gesture recognition," *World Academy of Science, Engineering and Technology*, vol. 49, no. 1, pp. 972–977, 2009.

[5] X. Zabulis, H. Baltzakis, and A. Argyros, "Vision-based hand gesture recognition for human computer interaction," in *The Universal Access Handbook*, Human Factors and Ergonomics, ch. 34, pp. 34.1 – 34.30, Lawrence Erlbaum Associates, Inc. (LEA), June 2009.

[6] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced computer vision with microsoft kinect sensor: A review," *Cybernetics, IEEE Transactions on*, vol. 43, pp. 1318–1334, Oct 2013.

[7] L. Nanni, A. Lumini, F. Dominio, and P. Zanuttigh, "Effective and precise face detection based on color and depth data," *Applied Computing and Informatics*, vol. 10, no. 12, pp. 1 – 13, 2014.

[8] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–511, IEEE, 2001.

[9] P. Kakumanu, S. Makrogiannis, and N. Bourbakis, "A survey of skin-color modeling and detection methods," *Pattern Recogn.*, vol. 40, pp. 1106–1122, Mar. 2007.

[10] Z. Mo and U. Neumann, "Real-time hand pose recognition using low-resolution depth images," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, pp. 1499–1505, 2006.

[11] Z. Ren, J. Yuan, and Z. Zhang, "Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera," in *Proc. of the 19th ACM international conference on Multimedia*, MM '11, (New York, NY, USA), pp. 1093–1096, ACM, 2011.

[12] X. Liu and K. Fujimura, "Hand gesture recognition using depth data," in *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pp. 529–534, May 2004.

[13] E. Kollorz, J. Penne, J. Hornegger, and A. Barke, "Gesture recognition with a time-of-flight camera," *Int. J. Intell. Syst. Technol. Appl.*, vol. 5, pp. 334–343, Nov. 2008.

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

[15] M. Van den Bergh and L. Van Gool, "Combining rgb and tof cameras for real-time 3d hand gesture interaction," in *Applications of Computer Vision (WACV), 2011 IEEE Workshop on*, pp. 66–72, Jan 2011.

[16] A. Kurakin, Z. Zhang, and Z. Liu, "A real-time system for dynamic hand gesture recognition with a depth sensor," in *Proc. of EUSIPCO*, 2012.

[17] Z. Ren, J. Meng, and J. Yuan, "Depth camera based hand gesture recognition and its applications in human-computer-interaction," in *Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on*, pp. 1–5, Dec 2011.

[18] Y. Wen, C. Hu, G. Yu, and C. Wang, "A robust method of detecting hand gestures using depth sensors," in *Haptic Audio Visual Environments and Games (HAVE), 2012 IEEE International Workshop on*, pp. 72–77, 2012.

[19] F. Pedersoli, N. Adami, S. Benini, and R. Leonardi, "Xkin - extendable hand pose and gesture recognition library for kinect," in *In: Proceedings of ACM Conference on Multimedia 2012 - Open Source Competition*, (Nara, Japan), Oct. 2012.

[20] F. Pedersoli, S. Benini, N. Adami, and R. Leonardi, "Xkin: an open source framework for hand pose and gesture recognition using kinect," *The Visual Computer*, vol. 30, no. 10, pp. 1107–1122, 2014.

[21] G. Pozzato, S. Michieletto, E. Menegatti, F. Dominio, G. Marin, L. Minto, S. Milani, and P. Zanuttigh, "Human-robot interaction with depth-based gesture recognition," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems*, (Washington, DC, USA), pp. 379–383, July 2014.

[22] P. Suryanarayan, A. Subramanian, and D. Mandalapu, "Dynamic hand pose recognition using depth data," in *Proc. of Int. Conference on Pattern Recognition (ICPR)*, pp. 3105 –3108, aug. 2010.

[23] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, ACM Press, 1992.

[24] J. Wang, Z. Liu, J. Chorowski, Z. Chen, and Y. Wu, "Robust 3d action recognition with random occupancy patterns," in *Proceedings of the 12th European Conference on Computer Vision - Volume Part II*, ECCV'12, (Berlin, Heidelberg), pp. 872–885, Springer-Verlag, 2012.

[25] N. Pugeault and R. Bowden, "Spelling it out: Real-time asl fingerspelling recognition," in *Proceedings of the 1st IEEE Workshop on Consumer Depth Cameras for Computer Vision*, pp. 1114–1119, 2011.

[26] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] C. Keskin, F. K$#305;ra$#231;, Y. E. Kara, and L. Akarun, "Hand pose estimation and hand shape classification using multi-layered randomized decision forests," in *Proceedings of the 12th European Conference on Computer Vision - Volume Part VI*, ECCV'12, (Berlin, Heidelberg), pp. 852–863, Springer-Verlag, 2012.

[28] K. Biswas and S. Basu, "Gesture recognition using microsoft kinect," in *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pp. 100 –103, Dec. 2011.

[29] P. Doliotis, A. Stefan, C. McMurrough, D. Eckhard, and V. Athitsos, "Comparing gesture recognition accuracy using color and depth information," in *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments(PETRA'11)*, pp. 20:1–20:7, 2011.

[30] T. Wan, Y. Wang, and J. Li, "Hand gesture recognition system using depth data," in *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pp. 1063 –1066, April 2012.

[31] C. Sun, T. Zhang, B.-K. Bao, C. Xu, and T. Mei, "Discriminative exemplar coding for sign language recognition with kinect," *Cybernetics, IEEE Transactions on*, vol. 43, pp. 1418–1428, Oct 2013.

[32] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "Scape: Shape completion and animation of people," in *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, (New York, NY, USA), pp. 408–416, ACM, 2005.

[33] L. Sigal, A. O. Balan, and M. J. Black, "Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion," *Int. J. Comput. Vision*, vol. 87, pp. 4–27, Mar. 2010.

[34] I. Oikonomidis, N. Kyriazis, and A. Argyros, "Efficient model-based 3d tracking of hand articulations using kinect," in *Proceedings of the 22nd British Machine Vision Conference (BMVC 2011)*, aug. 2011.

[35] L. Ballan, A. Taneja, J. Gall, L. Van Gool, and M. Pollefeys, "Motion capture of hands in action using discriminative salient points," in *Proceedings of the 12th European Conference on Computer Vision - Volume Part VI*, ECCV'12, (Berlin, Heidelberg), pp. 640–653, Springer-Verlag, 2012.

[36] C. Keskin, F. Kirac, Y. Kara, and L. Akarun, "Real time hand pose estimation using depth sensors," in *ICCV Workshops*, pp. 1228 –1234, nov. 2011.

[37] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from

single depth images," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 1297–1304, IEEE, 2011.

[38] L. E. Potter, J. Araullo, and L. Carter, "The leap motion controller: A view on sign language," in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI '13, (New York, NY, USA), pp. 175–178, ACM, 2013.

[39] C. Guerrero-Rincon, A. Uribe-Quevedo, H. Leon-Rodriguez, and J.-O. Park, "Hand-based tracking animatronics interaction," in *Robotics (ISR), 2013 44th International Symposium on*, pp. 1–3, 2013.

[40] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2 ed., 2003.

[41] C. Dal Mutto, F. Dominio, P. Zanuttigh, and S. Mattoccia, "Stereo vision and scene segmentation," in *Current Advancements in Stereo Vision*, Human Factors and Ergonomics, ch. 2, Intech, July 2012.

[42] D. Herrera, J. Kannala, and J. Heikkilä, "Joint depth and color camera calibration with distortion correction," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 10, pp. 2058–2064, 2012.

[43] L. Bezze, C. Dal Mutto, P. Zanuttigh, F. Dominio, and G. M. Cortelazzo, "Tof cameras and microsoft kinect depth sensor for natural gesture interfaces," in *ACM CHItaly*, (Alghero, Italy), September 2011.

[44] C. D. Mutto, P. Zanuttigh, and G. Cortelazzo, "A probabilistic approach to tof and stereo data fusion," in *Proceedings of 3DPVT 10*, (Paris (France)), May 2010.

[45] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, "Analysis of the accuracy and robustness of the leap motion controller," *Sensors*, vol. 13, no. 5, pp. 6380–6393, 2013.

[46] B. J.Y., "Camera calibration toolbox for matlab."

[47] G. Bradski *Dr. Dobb's Journal of Software Tools*.

[48] L. A. Westover, "Splatting: A parallel, feed-forward volume rendering algorithm," tech. rep., Chapel Hill, NC, USA, 1991.

[49] M. Di Noia, "Combined use of color and depth information for hand gesture recognition," Master's thesis, Masters School of Telecommunications Engineering, Department of Information Engineering, Padova, 2013.

[50] G. Marin, M. Fraccaro, M. Donadeo, F. Dominio, and P. Zanuttigh, "Palm area detection for reliable hand gesture recognition," *Proceedings of MMSP*, vol. 2013, 2013.

[51] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, pp. 679–698, June 1986.

[52] A. W. Fitzgibbon and R. B. Fisher, "A buyer's guide to conic fitting," in *Proceedings of the 6th British Conference on Machine Vision (Vol. 2)*, BMVC '95, (Surrey, UK, UK), pp. 513–522, BMVA Press, 1995.

[53] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[54] F. Dominio, G. Marin, M. Piazza, and P. Zanuttigh, "Feature descriptors for depth-based hand gesture recognition," in *Computer Vision and Machine Learning with RGB-D Sensors* (L. Shao, J. Han, P. Kohli, and Z. Zhang, eds.), Advances in Computer Vision and Pattern Recognition, pp. 215–237, Springer International Publishing, 2014.

[55] F. Dominio, M. Donadeo, G. Marin, P. Zanuttigh, and G. M. Cortelazzo, "Hand gesture recognition with depth data," in *Proceedings of the 4th ACM/IEEE International Workshop on Analysis and Retrieval of Tracked Events and Motion in Imagery Stream*, ARTEMIS '13, (New York, NY, USA), pp. 9–16, ACM, 2013.

[56] F. Dominio, M. Donadeo, and P. Zanuttigh, "Combining multiple depth-based descriptors for hand gesture recognition," *Pattern Recognition Letters*, vol. 50, no. 0, pp. 101 – 111, 2014. Depth Image Analysis.

[57] G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with jointly calibrated leap motion and depth sensor," *Multimedia Tools and Applications (Accepted for publication).*

[58] G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with leap motion and kinect devices," in *Image Processing, 2014. ICIP 2014. IEEE International Conference on*, 2014.

[59] Wikipedia, "Curvature of plane curves."

[60] S. Manay, D. Cremers, B.-W. Hong, A. Yezzi, and S. Soatto, "Integral invariants for shape matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 1602 –1618, October 2006.

[61] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. Lopez, and J. Soares, "Leafsnap: A computer vision system for automatic plant species identification," October 2012.

[62] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America A*, vol. 4, no. 4, pp. 629–642, 1987.

[63] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893 vol. 1, June 2005.

[64] L. Nanni, A. Lumini, F. Dominio, M. Donadeo, and P. Zanuttigh, "Combination of depth and texture descriptors for gesture recognition," in *Advances in Machine Learning Research* (S. Shandilya, ed.), Engineering Tools, Techniques and Tables, Nova Science Publishers, 2014.

[65] V. Ojansivu and J. Heikkilä, "Blur insensitive texture classification using local phase quantization," in *Proceedings of the 3rd International Conference on Image and Signal Processing*, ICISP '08, (Berlin, Heidelberg), pp. 236–243, Springer-Verlag, 2008.

[66] L. Nanni, A. Lumini, F. Dominio, M. Donadeo, and P. Zanuttigh, "Ensemble to improve gesture recognition," *International Journal of Automated Identification Technology (to appear)*, 2014.

[67] X. Tan and B. Triggs, "Enhanced local texture feature sets for face recognition under difficult lighting conditions," in *Proceedings of the 3rd International Conference on Analysis and Modeling of Faces and Gestures*, AMFG'07, (Berlin, Heidelberg), pp. 168–182, Springer-Verlag, 2007.

[68] X. Tan and B. Triggs, "Enhanced local texture feature sets for face recognition under difficult lighting conditions," in *Proceedings of the 3rd International Conference on Analysis and Modeling of Faces and Gestures*, AMFG'07, (Berlin, Heidelberg), pp. 168–182, Springer-Verlag, 2007.

[69] T. Ojala, M. Pietikinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern Recognition*, vol. 29, no. 1, pp. 51 – 59, 1996.

[70] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[71] S. Knerr, L. Personnaz, and G. Dreyfus, "Single-layer learning revisited: a stepwise procedure for building and training a neural network," in *Neurocomputing* (F. Souli and J. Hrault, eds.), vol. 68 of *NATO ASI Series*, pp. 41–50, Springer Berlin Heidelberg, 1990.

[72] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011.

[73] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[74] L. Nanni, S. Brahnam, C. Fantozzi, and N. Lazzarini, "Heterogeneous ensembles for the missing feature problem," April 2013.

[75] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, "On combining classifiers," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, pp. 226–239, Mar. 1998.

[76] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, pp. 832–844, Aug. 1998.

[77] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, pp. 832–844, Aug. 1998.

[78] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, pp. 1619–1630, oct 2006.

[79] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, no. 6, pp. 559–572, 1901.

[80] C.-X. Zhang and J.-S. Zhang, "Rotboost: A technique for combining rotation forest and adaboost," *Pattern Recogn. Lett.*, vol. 29, pp. 1524–1536, July 2008.

[81] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, pp. 119–139, Aug. 1997.

[82] L. Nanni, S. Brahnam, A. Lumini, and T. Barrier, "Data mining based on intelligent systems for decision support systems in healthcare," in *Advanced Computational Intelligence Paradigms in Healthcare 5* (S. Brahnam and L. Jain, eds.), vol. 326 of *Studies in Computational Intelligence*, pp. 45–65, Springer Berlin Heidelberg, 2011.

[83] X. He, D. Cai, S. Yan, and H.-J. Zhang, "Neighborhood preserving embedding," in *Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2*, ICCV '05, (Washington, DC, USA), pp. 1208–1213, IEEE Computer Society, 2005.

[84] L. I. Smith, "A tutorial on principal component analysis."

[85] Y. wei Chen, "Combining svms with various feature selection strategies," in *Taiwan University*, Springer-Verlag, 2005.

[86] Y.-W. Chen and C.-J. Lin, "Combining svms with various feature selection strategies," in *Feature extraction*, pp. 315–324, Springer, 2006.

[87] D. Aha and R. Bankert, "A comparative evaluation of sequential feature selection algorithms," in *Learning from Data* (D. Fisher and H.-J. Lenz, eds.), vol. 112 of *Lecture Notes in Statistics*, pp. 199–206, Springer New York, 1996.

[88] T. Fawcett, "Roc graphs: Notes and practical considerations for researchers," tech. rep., 2004.

[89] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006.

[90] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[91] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Mach. Learn.*, vol. 51, pp. 181–207, May 2003.