UNIVERSITÀ DEGLI STUDI
DI PADOVA

Dipartimento di Ingegneria dell'Informazione - DEI
Corso di Laurea Magistrale in Ingegneria Informatica

An Integer Programming Approach to Hand Pose Estimation Using Depth Data

Laureando: Ludovico Minto

Relatore:
Prof. Pietro Zanuttigh

Correlatore:
Dr. Giulio Marin

Anno Accademico 2013-2014

Abstract

Hand pose estimation and gesture recognition is one of the most challenging problems in the field of Computer Vision. This is mainly due to the large variability in hand shape and finger position, which can be often complicated by the presence of various types of occlusions. In addition, stringent time requirements have to be met in order to make any proposed scheme feasible for real time applications. Up to now, many approaches have been explored in order to address the problem, but a general solution that fits well in all cases is still to be found. This thesis introduces a novel approach to tackle the problem of estimating the pose of an hand from a single depth frame. The entire process never uses color information, nor it relies on the knowledge of previous frames in order to improve its estimates. We will come up with a scheme for finger and palm regions extraction, followed by a Mixed Integer Programming formulation to complete the process of finger segmentation. A key idea is a novel method which is able to rearrange the hand point cloud, through a simple transformation, in a way that better fits for recognizing its structure. A certain generality is preserved throughout all steps, leaving the basic framework of the approach quite independent from specific characteristics of the hand shape. This last aspect gives room for further improvements as well as for possible applications to different scenarios.

Keywords

Depth, Hand Pose Estimation, Fingers Recognition, Palm Recognition, Integer Linear Programming, Clustering, 3D Segmentation.

Contents

List of Figures

Chapter 1

Introduction

Automated gesture recognition is a fundamental step toward a new concept of human-machine interaction. Gesture-based communication is likely to gain more and more relevance in the near future, possibly overcoming some of today most popular input methods, from keyboard and mouse to later touch technologies. One of the main advantages over more traditional interfaces is in that no physical contact is required between the user and the machine, thus widening significantly the scope of applicability of gesture-based systems. Gesture recognition technologies experienced a great boost only in the very recent years, pushed by the enthusiastic acceptance they received since the beginning in the video-game and entertainment industry [1]. A consequent rapid development of low cost 3D data acquisition technologies has given an easy access to depth information, thus allowing a representation of captured objects much more close to reality. Nowadays, improvements in this field are all but slowing down, and the recent evolution of consumer TOF sensors is making available an even more accurate and efficient way to retrieve depth information. Nevertheless, some of traditionally difficult problems in the domain of Computer Vision, such as the one treated in this thesis, are still challenging researchers.

The problem of hand pose estimation and gesture recognition is one of the harder to be faced, due to the large variability in hand shape and finger position, which can be often complicated by the presence of various types of occlusions. Up to now, many approaches have been considered in order to address the problem, giving birth in most of cases to a plethora of byzantine solutions, which behave well under very restrictive conditions but lacks in generality. Various attempts has also been made in order to circumvent the problem, by focusing much more on the final objective, i.e. hand gesture

[1]One of the most notably examples of this success was Microsoft Kinect sensor for Xbox 360, with up to 24 millions units sold since its launching in 2010.

recognition, and trying to achieve the goal by using general tools such as machine learning techniques. Most of the time, the quality of the solution provided by these approaches heavily relies on the goodness of features that are taken into account. Clearly, the more information we can collect about the hand and fingers position, the better we can extract features that can be successfully used.

In this thesis a new way is presented to handle the problem of static hand pose estimation. The approach is kept as much general as possible so that further developments may be easily designed and tested. To this end, some hints are given in Chapter 9. The entire process can be structured as a multi-staged pipeline. In the first stage raw data is collected from the depth sensor, and stored as a point cloud of the captured scene. Points in the scene that are not part of the hand are discarded. Retained points are then passed to the second stage where they are filtered using a Statistical Outlier Removal Filter and a Bilateral Filter, to remove noise and to perform a light smoothing respectively. In the third stage Normal Guided Contraction method presented in 4 is applied to the point cloud so that a new cloud is produced. In the next stage the original cloud and the transformed one are compared, and a label is assigned to each point classifying it as a finger or a palm point. A refinement of the labels is also required in order to in order to increase the confidence of their prediction. The last two stages together are devoted to finger classification. In the fifth stage finger points are clustered to obtain an over-segmentation of the finger region. In the last stage, a Mixed Integer Programming model is solved in order to group the clusters into five super-clusters, each one corresponding to a finger.

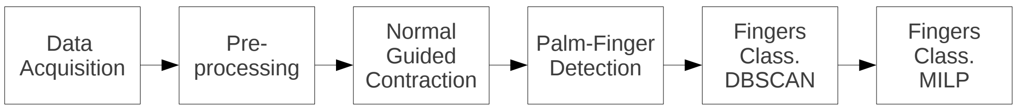

Figure 1.1: Pipeline of the proposed approach

1.1 Related Works

Hand pose estimation is a long-lived problem in the field of Computer Vision, and it has been intensely studied during the past two decades. A lot of approaches have been proposed throughout the years, starting from single source video-based solutions, up to the most recent methods which exploits multiple-view systems and depth information coming from range sensors. One of the earliest relevant works is [1], where the pose of an hand is recovered from two camera gray-scale image sequences. A three dimensional 27 DOF[2] kinematik system is used to model the hand, and a set of features is chosen so that their values are tightly linked with the pose of the hand. Then an estimate of the state of the hand is computed by inverting the model so that it best fits the set of observed features. Indeed, this is the basic mechanism that guide model-based approaches, where the hand pose is recovered by matching the projection of a 3D model against a set of measured image features. An extensive review of most substantial works that appear up to the recent past is given in [2].

The approach used in [3], where full body skeleton is estimated from a single depth frame, is also adopted by [4]. A 3D hand model is considered, made of 21 different parts. A per pixel classification based on Randomized Decision Forests is performed to assign each pixel of the depth frame to a hand part in the model. The labelled pixels are then exploited to compute joints locations for the hand model. Hough forests and particle swarm optimization are instead used in [5] and in [6] respectively. In [6], both RGB and depth channels are also exploited in order to improve performances and robustness. The approach illustrated in [7] takes advantage of temporal coherence constraints to perform dynamic hand pose estimation. Here a hand is modelled using a number of spheres, and the problem is formulated as a minimization problem. Gradient based and stochastic optimization methods are employed to achieve fast convergence and good accuracy.

The problem of hand gesture recognition is strictly tied to hand pose estimation, since solvers often use information on the pose of the hand in order to perform gesture recognition. Nevertheless, there are also many situations in which gesture recognition is achieved by directly exploiting low level features. Some approaches based on Support Vector Machines classification are presented in [8, 9, 10]. Another recent work is [11], where a special feature is used in order to discriminate extremal zones at various scales. Fingertip localization is performed using this feature and the output is fed to a k-Nearest Neighbors classifier for gesture recognition.

Commercial systems already exist for hand pose estimation and gesture recognition. Leapmotion [12] is one of the latest, and it is designed for close-range hand pose estimation.

[2]DOF, Degrees Of Freedom

1.2 Thesis Outline

The present thesis is organized following roughly the sequence of steps that form the entire processing pipeline.

The first stage, where raw data are collected from the range sensor, is illustrated in Chapter 2. Preprocessing filtering actions are described in Chapter 3, where Statistical Outlier Removal and the Bilateral filters are presented. Also Radius Removal filter is included here, even if it is applied only in later stages. In Chapter 4 the novel method of Normal Guided Contraction (NGC) is presented, after a brief description on how surface normals can be estimated for a point cloud. Direct applications of NGC will be considered in the subsequent two chapters. In Chapter 5 a method is described to extract palm and fingers points from the hand point cloud, thus performing a first-level palm-finger segmentation. First results for hand pose estimation will be achieved here. Finally, chapters 6 and 7 account for a second-level segmentation. Chapter 6 will illustrate how to get a finer segmentation than the simple palm-fingers partitioning calculated in Chapter 5. The clustering algorithm DBSCAN will be applied in order to group together points belonging to the same finger. No general configuration of DBSCAN parameters will be found to correctly perform this goal, and two possible ways to achieve approximate finger segmentation will be proposed. In Chapter 7 a Mixed Integer Linear Programming model will be introduced in order to come up with a correct labelling of finger points, using the partial results obtained in the previous chapter.

Chapter 8 will expose testing results, while in Chapter 9 a general review of the thesis will be discussed, as well as some hints for possible future continuations.

Chapter 2

Depth Data Acquisition

I begin here my discussion by considering how raw data are collected. Of course, this can be regarded as the first stage of the whole process pipeline, and can be treated separately from the subsequent analysis and evaluation steps, which will be examined in the next chapters. As I chose to base all my computations and results on depth data, how color information is acquired will not taken into account, nor it will bother how color should be properly aligned with depth samples. Of course, there is a plenty of works showing how to take advantage of RGB data in order to obtain better results. As an example, it is worth mentioning [13], where a robust denoising algorithm is presented for outlier detection and smoothing of coloured depth data. This kind of results can be particular useful to enhance the quality of local features such as surface normals, which in turn can positively affect the performances of some of the methods that will be used in this work, like Normal Guided Contraction presented in Chapter 4. However, it is beyond the scope of this thesis to evaluate the many improvements and clever solutions that can be applied at this stage of the process by considering visual data.

In section 2.1 I will briefly introduce the physical mechanisms that underneath the functioning of TOF sensors, which represent a large segment of today consumer range cameras, such as Intel Creative Senz3D, the camera that has been used during the experimentations. In section 2.2 will then be presented the equipment that has been used to collect depth information while section 2.3 will be devoted the dataset that has been created to evaluate all solutions.

2.1 TOF Cameras

I briefly illustrate here the fundamental working principle of Time-of-Flight 3D technology, which is the same technology used by SoftKinetic DS325 to retrieve depth data. Incidentally, DS325 is also the depth sensor which is embedded in the Intel Creative Senz3D [14], the camera that has been used for the tests. Beside TOF, there also exists other types of technologies which are inherently different, yet are successfully adopted by a large segment of range cameras. Structured Light is one of them: it basically consists in projecting a well known and structured pattern of light over the observed objects, and then compute the distances starting from the way the pattern is distorted. As an example, a grid of infrared spots is used by the first generation Microsoft Kinect [15] sensors.

Time-of-flight cameras, shortly referred to as TOF cameras, is a specific type of range camera which tries to estimate distances by measuring the round trip time spent by a light pulse to "flight" - hence the name TOF - from the camera to a point of the image, and to be reflected back. A modulated light source such as a laser or LED is used in order to distinguish the pulse from the light generated by other sources. The reflected back light is received by the camera and detected by an array of pixels.

The phase delay φ_p between the original signal and the signal detected by pixel p, which corresponds to the signal sent back by some real world point x_p, is then computed through a series of processing steps. Finally, the distance from the camera to x_p, which is proportional to φ_p, is easily recovered.

More in detail, let $s(t) = \sin(2\pi f_m t)$ be the transmitted light signal, where f_m is the modulation frequency, and let $r_p(t) = R_p \sin(2\pi f_m - \varphi_p) + I_p$ be the signal received by pixel p after demodulation, where I_p is an offset accounting for background illumination. The phase shift φ_p can be recovered by considering four samples s_r^0, s_r^1, s_r^2, s_r^3 taken from $r_p(t)$ at four $\pi/2$ phase intervals, and then using the following relation:

$$\varphi = \arctan(\frac{s_r^3 - s_r^1}{s_r^0 - s_r^2}) \tag{2.1}$$

Finally, let c be the speed of light, the distance d_p from the camera to the point x_p can be unambiguously calculated, under the assumption $d_p < \frac{1}{2}\frac{c}{f_m}$, as follows:

$$d_p = \frac{1}{2} \cdot \frac{\varphi_p}{2\pi} \cdot \frac{c}{f_m} = \frac{c}{4\pi f_m}\varphi_p \tag{2.2}$$

Moreover, the values of R_p and I_p are also computed, in order to get an estimate of SNR and a measure of distance uncertainty. R_p and I_p are obtained from the four samples using the following formulas:

$$R_p = \frac{\sqrt{(s_r^2 - s_r^0)^2 + (s_r^3 - s_r^1)^2}}{2} \tag{2.3}$$

$$I_p = \frac{s_r^0 + s_r^1 + s_r^2 + s_r^3}{4} \tag{2.4}$$

2.2 Intel Creative Senz3D

Intel Creative Senz3D [14] is the camera that has been used to acquire depth data and perform all experimentations. Other two alternatives, first generation Microsoft Kinect [15] and Leap Motion Controller [12] were considered. Even if Kinect sensor has a depth resolution comparable to the one of Senz3D and a wider operational range, I finally opted for the Senz3D which seemed to return better and much less noisy measures in the close range. I excluded also the use of Leap Motion Controller, since no SDK for it was available at the time. Intel Creative Senz3D, which was sold starting from the third quarter of 2013, uses the TOF depth sensor DS325 from SoftKinetic [16]. Its best depth resolution is QVGA 320×240, while its suggested depth range goes from 0.15 meters up to 1 meter, which can be considered optimal for the kind of acquisitions we wanted to carry out. In order to get access to the raw data from the depth stream, function utilities were used as supplied by the Intel Perceptual Computing SDK 2013 [17]. The depth frames captured by the sensor revealed to be quite accurate, and at the camera showed to be one of the best suited for my purposes.

2.3 The Dataset

In order to evaluate the quality of results, a small dataset has been set up by collecting depth frames of various gestures, each one performed for a certain number of repetitions, changing for each repetition both the hand in-plane orientation and hand plane inclination with respect to the camera. The dataset comprises an overall of 22 gestures, 8 repetitions each, from a single subject. It is far from being considered exhaustive, yet it demonstrated to be quite effective in showing all the limits of some bad choices that were taken during the research, guiding the work towards increasing improvements.

All gestures were captured in standard environment conditions, at a distance of about 0.4 to 0.5 meters from the camera. A wall was used as an invariant background for all captured scenes, always keeping a fixed distance of no more than 1 meter between the wall and the sensor. This also partially diminished the number of outliers which usually comes out when the background is positioned outside the TOF sensor range. In these cases, a number of border points belonging to objects inside the range may also be affected, since their depth values are averaged with the wrong depth values estimated for background points that lay behind. Anyway, an appropriate filter such as the Statistical Outlier Removal filter discussed in section 3.2, would be able to address the problem and quite easily remove most part of noisy points. Single depth frames were recorded and stored, one for each gesture repetition, at a resolution of 320×240, along with the corresponding RGB frames from the color stream. The latter were saved for visualization purposes only, since, as I have already mentioned, all presented procedures never use color information. For each frame, a point cloud belonging to the hand is extracted, and stored to be used as a basic input for all successive steps.

The extraction of hand points was performed following a process similar to the one used in [18]. More sophisticated methods for hand tracking can be exploited, such as the one proposed [19]. Another approach is suggested in [11], where the Oriental Radial Distribution descriptor is used at hand scale in order to discriminate hand region from other parts of the body. Nevertheless, I will not enter now a more detailed explanation of how this task was achieved, being marginal with respect to the whole discussion. In the following chapters, it should never be considered an issue how hand points are taken from the scene, and all methods will always deal with clouds containing hand points only.

Aside from the very basic set of hand samples collected for this thesis, it has also been evaluated the possibility to test results on much larger and comprehensive datasets, such as the MSRA Hand Tracking Database presented in [7], which uses Intel Creative Senz3D. It certainly happens to be one of the most richer and well documented datasets publicly available at this moment, collecting a great variety of challenging poses along with their corresponding ground truth. However, as it is mainly conceived as a test-bed for dynamic hand tracking, a considerable fraction of poses included in it can hardly be estimated by static approaches as the one presented in this thesis and. On the contrary, they should be evaluated by taking advantage of the high correlation existing between poses in temporal neighbour frames. Moreover, all the clouds in the dataset are given after being passed through some form of preprocessing, which is likely not the most appropriate for this thesis methods to run properly.

Another rich and well organized dataset is ColorTip Dataset, used in [11] and created by employing a first generation Microsoft Kinect sensor.

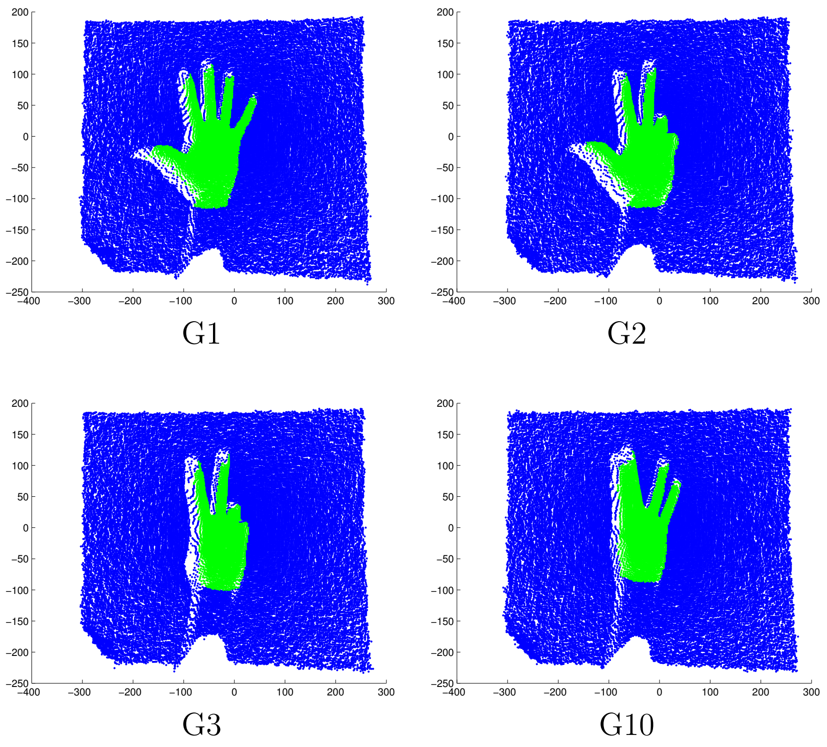

Figure 2.1: Captured scenes: hand points shown in green

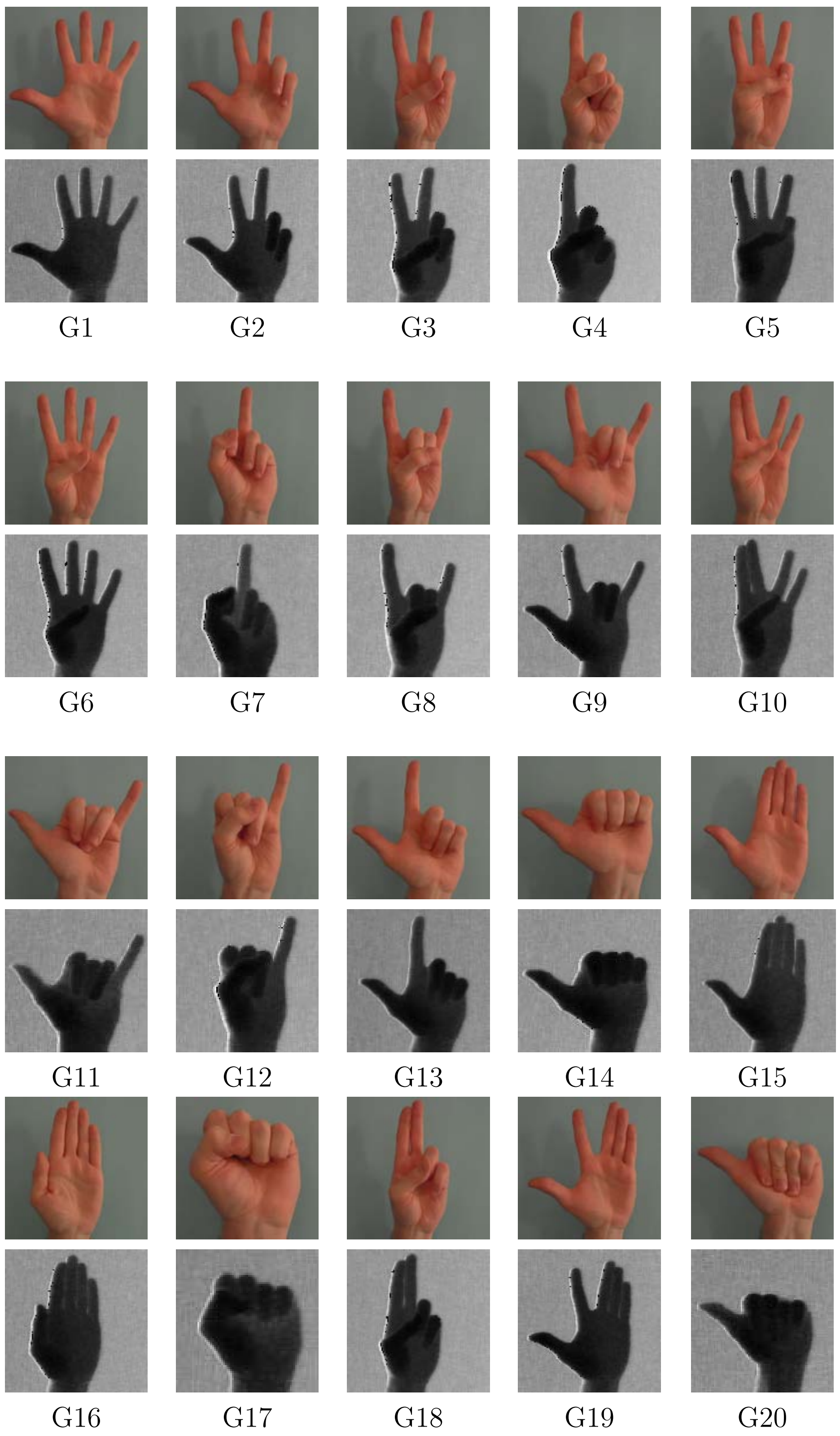

Figure 2.2: Sample images and depth maps for each of the gestures in the database.

However, I dropped the possibility to perform tests on this dataset since frames in it are taken at a full body scale and clouds associated to the hands are not as much dense and accurate to be successfully fed as input to our procedures.

Chapter 3

Pre-processing

Some essential filtering and refinement procedures are included in this chapter. They are applied to raw data in order to end up with a cleaner and accurate description of the observed reality. The ultimate goal of this stage is to produce a three dimensional PCD approximating, possibly with a good accuracy, the hand surface as it is captured by means of the range sensor.

Once raw data have been acquired, a series of filtering transformations are needed before proceeding with further and more sophisticated manipulations. Indeed, the effectiveness of a large number of well known local descriptors is highly influenced by the grade of accuracy shown by the PCD in approximating the hand surface at a fine-grained level. The estimation of surface normals for each point of the PCD, which underneath much of the methods proposed in the following chapters, follows this rule: the better the depth samples are able to approximate the neighbourhood of a given point, the better would result the evaluation of the surface normal at that point. A good quality PCD is particularly desirable in order to avoid to come up with poor solutions when applying the NGC method in Chapter 4. On the other side, as we will see, using these filtering tools with improperly tuned parameters could result both in an excessive point removal and in an excessive flattening of all surfaces, thus rapidly worsening the performances of NGC. In particular, when using the Bilateral Filter in section 3.3, it should pay attention to preserve the relevant curvatures of the cloud, especially finger curvature and their tubular shape.

In section 3.1 a simple filter will be presented which aims to remove isolated points from the cloud. The Statistical Outlier Removal Filter in section 3.2 has nearly the same function, but it is able to somehow account for the specific distribution of the points in the cloud, and consequently adapt its inner removal parameters. Finally, the Bilateral Filter is illustrated in the last section.

The Point Cloud Library To perform the filtering stage, as well as to compute the surface normals in section 4.1, a set of appropriate functions from the open-source Point Cloud Library (PCL) [20] have been used. For the purpose of this work, only a few of them, taken from the filtering and feature extraction sections of the library, were sufficient. However, many other set of functions are included in PCL, accounting for efficient solutions about key-points extraction, PCD segmentation, surface reconstruction as well as for many other crucial tasks in 3D processing. The PCL project started its development in 2010, and it counts, at these days, a significant number of implementations of state-of-the-art algorithms in the field of point cloud processing. The code is freely downloadable from `http://pointclouds.org/`.

3.1 Radius Outlier Removal Filter

Radius Outlier Removal, as implemented in PCL, is a quite simple filter. For each point $\mathbf{p}$ in the cloud $\mathcal{P}$, a radius-based neighborhood $\mathcal{N}_r(\mathbf{p})$ is considered, containing all and only those point in the cloud whose distance from $\mathbf{p}$ is not greater than a given value r, that is:

$$\mathcal{N}_r(\mathbf{p}) = \{\mathbf{p}' \in \mathcal{P} \mid \|\mathbf{p} - \mathbf{p}'\|_2 \leq r\} \tag{3.1}$$

The filter removes all points $\mathbf{p}$ such that $\mathcal{N}_r(\mathbf{p}) < n_{min}$, where n_{min} is a parameter accounting for the minimum number of neighbors that a point should have to be not considered an outlier. The output filtered cloud is then:

$$\mathcal{P}_{ror} = \{\mathbf{p} \in \mathcal{P} \mid \mathcal{N}_r(\mathbf{p}) \geq n_m in\} \tag{3.2}$$

Having to specify both parameters r and n_{min} is a major drawback, however, since the filter do not account itself for changes in the average density of the input clouds. Clearly, a point $\mathbf{p}$ for which $\mathcal{N}_r(\mathbf{p}) < n_{min}$ should not be considered an isolated point if for all other points in the cloud holds the same too. The filter presented described in the nex section is able to fix this issue.

3.2 Statistical Outlier Removal Filter

The Statistical Outlier Removal filter, as it is presented in [21], relies on the assumption that outlier points are those points whose number of neighbors deviate for a certain amount from the average number of neighbors characterizing points in the cloud. The Statistical Outlier Removal differs from more traditional approaches in that it does not require to explicitly specify the radius of the neighborhood, nor a minimum number of neighbors for a point to stay in the cloud.

More in detail, let $\mathbf{p}$ be a point in a cloud $\mathcal{P}$, and let $\mathcal{N}_k(\mathbf{p})$ be its k-neighborhood, that is the set of the k points in $\mathcal{P}$ closest to $\mathbf{p}$. The filter scheme works as follows: first the mean distance $\bar{d}_k(\mathbf{p})$ from $\mathbf{p}$ to its k neighbors in $\mathcal{N}_k(\mathbf{p})$ is computed for each point $\mathbf{p}$. Then a distribution on the values in the set of all mean distance values $\{\bar{d}_k(\mathbf{p}) \mid \mathbf{p} \in \mathcal{P}\}$ is calculated, together with an estimate of its mean and standard deviation, respectively μ_k and σ_k. The set of points in $\mathcal{P}$ which are retained is defined as:

$$\mathcal{P}_{sor} = \{\mathbf{p} \in \mathcal{P} \mid \mu_k - \alpha \cdot \sigma_k \leq \bar{d}_k(\mathbf{p}) \leq \mu_k + \alpha \cdot \sigma_k\} \tag{3.3}$$

where α is a positive real parameter that determines how much strong the removal action of the filter would be. The set of outliers is thus defined as $\mathcal{P}_o = \mathcal{P} \setminus \mathcal{P}_{sor}$.

The filter proved to be both effective and robust to variations in the average density of the hand PCDs that have been tested. This should not surprise since the filter ability to implicitly change its definition of outlier point together with changes in the statistical mean density characterizing the cloud. The filter has been applied to the raw PCDs as they were returned by the acquisition stage[1]. Many different values have been assigned to the input parameters k and α and then tested. The best results were achieved by setting $k = 100$ and $\alpha = 1.0$. It has to be stressed that, given the adaptive nature of the filter, the tuning of the two input parameters is quite easy, and it is rather unlikely to come up with unexpected results due to a bad setting. By comparison, as it will be evident in Chapter 6, one of the weaker points of DBSCAN algorithm is an intrinsic difficulty in selecting an appropriate set of values for its input parameters.

3.3 Bilateral Filter

In order to smooth the surface of each PCD, as a preprocessing step before computing the surface normals in section 4.1, a Bilateral Filter [22] is applied to the cloud returned as output by the Statistical Outlier Removal Filter described in the previous section.

The Bilateral Filter was originally conceived for smoothing 2D images while preserving edges. For simplicity, the discussion will be limited to 2D greyscale images. The idea is to consider two types of closeness for pixels

[1]Remember that it is always assumed that a mask has been applied to each PCD before it leaves the acquisition stage, so that only points belonging to the hand are retained

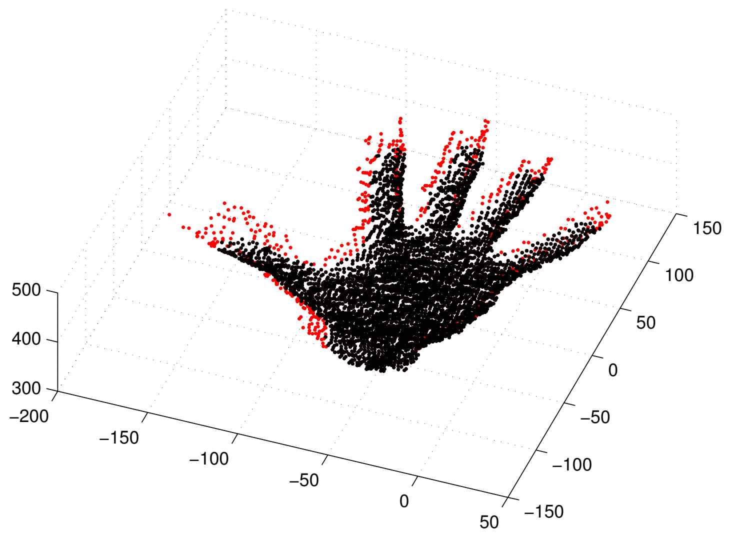

Figure 3.1: Statistical Outlier Removal Filter applied to G1: outliers in red

in the image, the usual space closeness for which two pixels are close one another if they are located in nearby positions, and an intensity closeness, for which two pixels are close as long as their grey levels are similar. The filter aims to perform a domain averaging, the way traditional smoothing filters do, with weights that depends both on space and intensity closeness. Basically, the gray level of each pixel x in the filtered image is computed as a weighted average of the gray levels of nearby pixels in the original image, with weights decaying so that pixels which are far from x both in range or intensity weight less than closer pixels. Denoting with $t(x)$ the gray level value of pixel x in the original image, then its corresponding gray level $t'(x)$ in the filtered image is computed as:

$$t'(x) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} c(\xi, x) s(t(\xi), t(x)) d\xi \tag{3.4}$$

where c is a function measuring the spatial closeness between x and a nearby point ξ, while s is a function that measures the similarity between the intensity levels $t(x)$ and $t(\xi)$ of pixels x and ξ respectively. In particular, for a Gaussian Bilateral Filter c and s are defined as two Gaussian window functions. More precisely we have:

$$c(\xi, x) = e^{-\frac{1}{2}(\frac{d(\xi,x)}{\sigma_s})^2} \tag{3.5}$$

where $d(\xi, x)$ is the Euclidean distance between ξ and x, and:

$$s(t(\xi), t(x)) = e^{-\frac{1}{2}(\frac{\delta(t(\xi),t(x))}{\sigma_r})^2} \tag{3.6}$$

where $\delta(t(\xi), t(x))$ is a function accounting for the distance between intensity values $t(\xi)$ and $t(x)$.

As pointed out in [22], the filter can be easily applied once we are able to define a range distance and an intensity distance between pixels in the image. It is then straightforward to extend its use to color images. The same way, it can be used on 3D points clouds.

An implementation is available for 3D point clouds in the PCL library. To be used, it requires an intensity level to be specified for each point in the cloud.

can be applied once an intensity level for each point is specified.

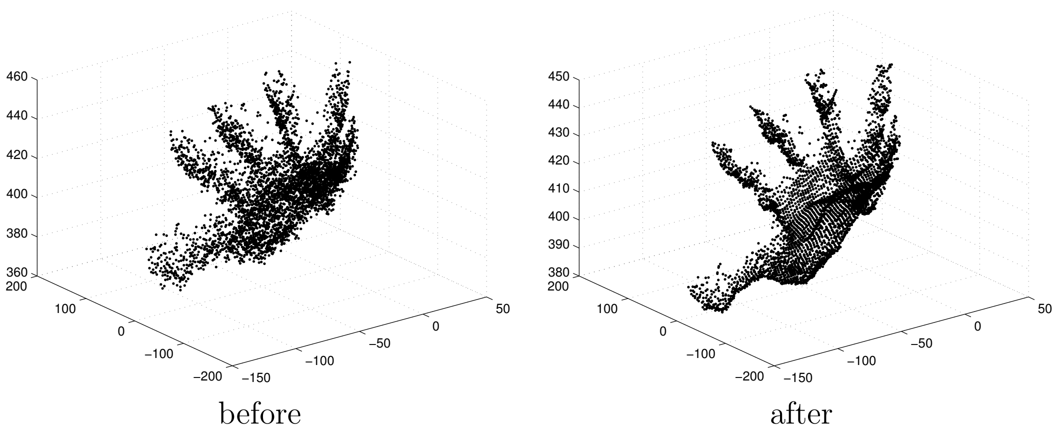

Figure 3.2: Bilateral Filter applied to G1.

Chapter 4

Normal Guided Contraction

Introduction

As discussed at the beginning, this thesis originated from the interest in exploring new ways to achieve hand pose estimation through a mathematical programming approach. In particular, it will be considered the problem of fitting a skeleton model of the hand to the real hand points captured by the camera, focusing on the possibility of reducing this problem to some generalized form of the well known Assignment Problem (AP). As we will see later in Chapter 7, the aim is that of assigning each joint of the skeleton model to a real point, provided that some additional constraints are met. One of the many obstacles that prevent this solution to be even barely acceptable is the excessive time required to perform computations. In spite of this, various tricks have been experimented in order slightly change the initial problem formulation and fasten its resolution. No matter how these little adjustments can be cleverly applied, the two factors which have, by far, the greatest impact on the time needed to find a solution are the number of variables present in the formulation and, partly, the number of constraints. To have an idea of the dimension this problem can take, consider the following. In an average practical case there are about from 1k to 5k real points or possible target, while the number of joints in the model to be assigned is usually set to 21. If no countermeasure is taken, whatever AP-like formulation we choose, the total number of variables in the model shall be comprised, more or less, from 20k up to 100k. This can be a big deal if we intend, as it is the case, to perform all computations in a fraction of a second.

To tackle this problem, tests were run only after heavily reducing the number of target points, by operating a random decimation or by dividing the space in a grid and taking the centroid of the points inside each square.

This trivial approach was able to resize down the problem by a factor of ten, however still not sufficient to achieve an adequate timing. On the other side, pushing forward the decimation ratio or by widening the grid squares resulted most of the time in poor quality results. Given this scenario, it is evident the necessity to perform a more intelligent selection of the target points. Of course, the ideal case is where the set of target points is reduced to exactly those points that an optimal algorithm would have chosen as the best targets for the joints of the skeleton model. An improvement in this sense can be achieved by keeping only a few target points for each finger, possibly choosing them so that they are equally spaced located along an hypothetical line running from the finger base to the finer tip.

It is clear that a key issue in order to implement this kind of reduction is to have access to some form of finger recognition, or even better segmentation. In particular, it happens to be of fundamental importance to be able not only to recognize those points belonging to the fingers from those ones belonging to the palm of the hand, but also to be able to differentiate points belonging to different fingers. With this purpose, a series of attempts have been carried out, finally leading to Normal Guide Contraction (NGC) method, which will be introduced in this chapter. Direct applications of NGC will instead be presented in Chapter 5, which can be seen as a natural prosecution of the present chapter.

4.1 Surface Normals Estimation

NGC makes direct use the surface normals that are associated to each point in the cloud. Moreover, its performances heavily relies on how well these normals are estimated. In general, surface normals are a crucial feature for many tasks in Computer Vision and Computer Graphics. They carry important information in that they locally describe the geometry of an object surface. Following [23], il will be shown here a simple way to compute them, which is based on Principal Component Analysis (PCA). This approach is also implemented by the PCL functions that have been used to estimate the surface normals of the hand PCD.

The problem of computing the surface normal of a point $\mathbf{p}$ placed on some object, can be restated as the problem of computing the unitary vector that describes the orientation of a plane tangent to the surface of the object in correspondence to $\mathbf{p}$. Let $\mathcal{N}_r(\mathbf{p})$ be the set of points in $\mathcal{P}$ whose distance from $\mathbf{p}$ is less than or equal to r, that is:

$$\mathcal{N}_r(p) = \{\mathbf{p}' \in \mathcal{P} \mid \|\mathbf{p} - \mathbf{p}'\|_2 \leq r\} \tag{4.1}$$

The plane tangent to the surface of $\mathcal{P}$ in $\mathbf{p}$ can be intuitively regarded as the plane that best fits the points inside $\mathcal{N}_r(\mathbf{p})$. A fast solution to this problem can be achieved by using PCA on the coordinates of the neighbor points in $\mathcal{N}_r(\mathbf{p})$, and looking for the component which accounts for the minimum variance. To do this, the centroid $\bar{\mathbf{p}}$ of the neighborhood $\mathcal{N}_r(\mathbf{p})$ is computed and then subtracted to each point $\mathbf{p}_k \in \mathcal{N}_r(\mathbf{p})$. A covariance matrix $C \in \mathbb{R}^{3\times3}$ is then computed on the re-centred points, so that:

$$C = \frac{1}{|\mathcal{N}_r(\mathbf{p})|} \sum_i (\mathbf{p}_k - \bar{\mathbf{p}}) \cdot (\mathbf{p}_k - \bar{\mathbf{p}})^T \tag{4.2}$$

Let v_j for $j = 1, 2, 3$ be the eigenvectors of C, and let λ_j for $j = 1, 2, 3$ be their corresponding eigenvalues. Observe that, since C is symmetric and positive semi-definite, then its eigenvalues are real numbers. If $0 \leq \lambda_1 \leq \lambda_2 \leq \lambda_3$, then $v_1/\|v_1\|$ is a solution to our problem, that is $v_1/\|v_1\|$ is orthogonal to the plane tangent to the surface of $\mathcal{P}$ in $\mathbf{p}$.

4.2 Normals Orientation

In the previous section the surface normal at a given point $\mathbf{p}$ in $\mathcal{P}$ has been computed by looking for a unitary length vector orthogonal to the plane tangent to the surface of $\mathcal{P}$ at $\mathbf{p}$. There is an intrinsic ambiguity in how a surface normal is defined since if a vector $\mathbf{n}$ is a unitary-length vector orthogonal to a plane, then also is $-\mathbf{n}$. Indeed, no guarantee exists about how the vector computed using Principal Component Analysis is oriented. Nevertheless, it is necessary for our purposes to point all surface normals toward the outside of the hand. It follows that it has to be found a way to flip in the right direction all normals that do not meet this requirement. In general this is not an easy task to perform. Luckily, this is not the case, since a single depth sensor is used. Given two possible choices $\mathbf{n}_i$ and $-\mathbf{n}_i$ to represent the surface normal at a point $\mathbf{p}_i \in \mathcal{P}$, the one which pointing toward the camera point-of-view $\mathbf{w}$ will be taken. It will then be sufficient to check if $\mathbf{n}_i$ satisfies the following equation:

$$\mathbf{n}_i \cdot (\mathbf{w} - \mathbf{p}_i) > 0 \tag{4.3}$$

If this condition holds then we are done, otherwise $-\mathbf{n}_i$ will be taken as the surface normal at $\mathbf{p}$.

4.3 Points Contraction

The idea, which is quite simple, is to move each point in the PCD toward the direction pointed by its surface normal, by a step whose magnitude has to be scaled depending on the target. Clearly, this makes sense if a pointing direction has been previously defined for each normal. As suggested in the previous section, all normals can be oriented so that they are all flipped toward the outside of the hand. In that case, if we look at the points enclosed in a neighborhood of a certain size, moving them along their normals would result in the points becoming closer and closer or more separated one another, depending on the surface curvature characterizing the neighborhood. In an hypothetical case where all points are samples of a sphere of a given radius r, and their normals are perfectly estimated and directed towards the inside of the sphere, choosing a step of magnitude r would come up with all points concentrated in the center of the sphere, as if a contraction of the original spherical structure takes place. On the contrary, if the normals were directed towards the outside, a sort of expansion would occur.

More formally, let $\mathcal{P} \subset \mathbb{R}^3$ be a PCD, such that each point is given as a vector whose components are its (x, y, z) coordinate values with respect to some reference system. Also, let $\mathbf{n}_p \in \mathbb{R}^3$ be the oriented surface normal associated to each point $\mathbf{p} \in \mathcal{P}$, calculated following sections 4.1 and 4.2. The transformation moves each point $\mathbf{p}$ into a new point $\mathbf{p}'$ such that:

$$\mathbf{p}' = \mathbf{p} + t \cdot \mathbf{n}_p \tag{4.4}$$

where t is a scalar parameter that determines the magnitude of the move. We let t to take negative values also, which accounts for a move in the opposite direction with respect to that one indicated by the normal.

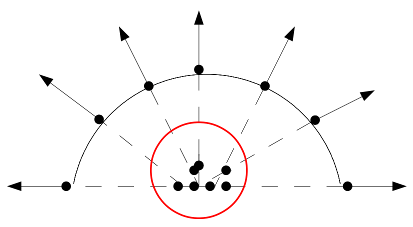

Figure 4.1: NGC: contraction using $t < 0$, in red high-density region.

NGC has its major impact for those points which are located on surfaces roughly approximating portions of a sphere of radius $|t|$, while it does not affect the relative position of points displaced on planar surfaces. By correctly choosing the sign of the step parameter t, the transformation would mainly contract structures within the PCD whose curvature radius nearly equals $|t|$, while leaving almost unchanged structures with a much greater curvature. Observe that, for points located in a neighborhood having a curvature $r < |t|$, a maximum in contraction is likely to be reached after moving the points at a distance of $|s|$ from their initial positions, while a new expansion would occur when for distance values in $(r, |t|]$. So, the overall effect would be a weaker contraction than for points in regions where the curvature is near $|t|$. By correctly choosing the sign of the step parameter t, the transformation would mainly contract structures within the PCD whose curvature is, more or less, equal to $|t|$.

Intuitively, this method gives us a way to transform the original cloud $\mathcal{P}$ in a new cloud $\mathcal{P}^{'}$ so that neighborhoods with a given curvature can be easily detected by looking for regions in $\mathcal{P}^{'}$ with higher density. Indeed, these regions approximatively correspond to parts of the cloud where the contraction was more effective.

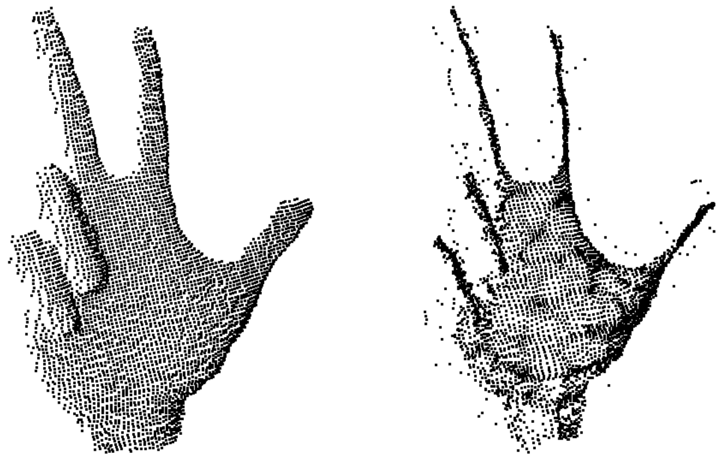

Figure 4.2: NGC applied to hand PCD: $t \simeq$ finger radius

Chapter 5

Palm and Fingers Extraction

In this chapter NGC is applied in order to operate a first low-level segmentation of the hand, partitioning the hand cloud $\mathcal{P}$ into two subsets $\mathcal{P}_f$ and $\mathcal{P}_p$, corresponding to the points belonging to the fingers and those belonging to the palm, respectively. The next chapter will handle the problem of separating points belonging to different fingers, to achieve an higher level segmentation. A simple palm-fingers segmentation can be still useful for a number of applications. Various methods can directly take advantage of this kind of information to achieve better results. Palm-fingers segmentation can be employed to enhance performances of methods implementing hand plane estimation through RANSAC-like fitting algorithms. In this case, a plane fitting could be carried out considering only points in $\mathcal{P}_p$, rather than the whole cloud. In turn, schemes such the one presented in [5] can potentially benefit of a better hand plane estimation when calculating the in-plane orientation of the hand. An affordable hand plane estimation can be an advantage also for palm-finger segmentation itself, since an higher quality partition can be attained by adding to $\mathcal{P}_f$, among other points, all points that are located at a distance from the plane greater of some threshold. In the following section a method will be presented to perform a first palm-finger labelling, by inspecting the transformed cloud obtained through NGC transformation. The second section will describe how to refine this labelling to get a more accurate partitioning.

5.1 A NGC-based partitioning

Here is presented a method to extract $\mathcal{P}_f$, the set of finger points, from $\mathcal{P}$, the point cloud of the hand. The palm points $\mathcal{P}_p$ will be simply defined as the set of points in $\mathcal{P}$ that are not labelled as finger points, that is $\mathcal{P}_p = \mathcal{P} \setminus \mathcal{P}_f$. The method relies on NGC transformation, and in particular searches for finger points by locally comparing the original point cloud $\mathcal{P}$ with its transformed counterpart $\mathcal{P}^c$.

As it has already mentioned in 4, the main contribution of NGC is that regions in $\mathcal{P}$ whose underneath structure has a curvature radius similar to the magnitude step of contraction are likely to be contracted in smaller areas than other regions do. The key observation is that fingers all share a similar tubular shape and an almost constant curvature radius, while the palm region is characterized by having a planar structure, or at least by having a local curvature way larger than the mean radius of a finger. In spite of this, a good choice for the NGC step parameter is to set its magnitude equals the average radius of a finger. As a consequence of NGC transformation, finger points will mostly appear in $\mathcal{P}^c$ inside areas of highest density, while remaining points, located in areas whose density is nearly left unchanged with respect to the original cloud, will roughly correspond to palm points. To be clear, we refer to the density of an area as the number of points that appear in that area. A way to estimate whether a given point $\mathbf{p}$ is part of a high or low density region is to count the number of points in the cloud whose distance from $\mathbf{p}$ is not greater than a certain value. Another possibility would be that of computing the mean distance from $\mathbf{p}$ to each point in its k-neighborhood. In this case, the smaller is the mean distance, the higher is the density of the cloud in the neighborhood of $\mathbf{p}$.

Let f_{ngc} be the transformation as defined in section 4.3, that is:

$$\begin{array}{rccc} f_{ngc} & : & \mathbb{R}^3 & \longrightarrow & \mathbb{R}^3 \\ & & \mathbf{p} & \longmapsto & \mathbf{p} + t \cdot \mathbf{n}_p \end{array} \tag{5.1}$$

The first and most obvious way to extract finger points would be then to look at $\mathcal{P}^c = f_{ngc}(\mathcal{P})$ and then to partition the cloud in two subsets $\mathcal{P}^c_H$ and $\mathcal{P}^c_L$, including respectively points in high-density areas, and points in the remaining low-density areas. We then set:

$$\mathcal{P}_f = \{p \in \mathcal{P} \mid f_{ngc}(p) \in \mathcal{P}^c_H\} \tag{5.2}$$

$$\mathcal{P}_p = \{p \in \mathcal{P} \mid f_{ngc}(p) \in \mathcal{P}^c_L\} \tag{5.3}$$

A critical step in this process is how to conveniently set the density threshold that discriminates between higher density values from lower ones. One way is to associate to each point $\mathbf{p}$ in $\mathcal{P}^c$ a density value computed as the number of neighbors inside a fixed range from $\mathbf{p}$. The average density is then computed for all points. Finally, all those points whose associated density is greater than the average are put in $\mathcal{P}^c_H$. During tests, this trivial approach was found to achieve better results than performing a 2-means on density values.

A second approach is illustrated here to extract finger points, which was found to behave better than the previous one in almost all situations. It can be observed that in certain cases, small strings of highly packed points arise in the transformed cloud even inside the palm region. This is particularly evident when the palm is arching, and folds appears on the palm surface. This unexpected behaviour is explained by the fact, that in correspondence to these folds, the local curvature of the cloud decreases and become more similar to the curvature of fingers. Moreover, since there are usually more points on the surface of the palm than on fingers, it would suffice a larger curvature to reach, after the contraction, the same point densities as those of fingers. To address this problem, a solution is to partition $\mathcal{P}$ by considering how the density values associated to the points in the cloud change before and after the contraction. In particular, for each point we compute the ratio between the density associated to the point in $\mathcal{P}$ and its density in $\mathcal{P}^c$. Also, we calculate the first density with respect to a larger neighborhood than the one used for the second density. Finally, finger points are extracted by taking points with a ratio higher than the average.

5.2 Refinements

Even if the method illustrated in the previous section gives a good estimate for $\mathcal{P}_f$ and $\mathcal{P}_p$, there may still be small spots of points labelled as finger points inside the palm region and, vice versa, there may be points labelled as palm points which actually belong to a finger. In order to refine the partitioning and remove these kind of mistakes, we operate a majority-voting re-labelling. We take each point $\mathbf{p}$ in the cloud $\mathcal{P}$ and look at its neighborhood $\mathcal{N}_r(\mathbf{p})$. If the $\mathcal{N}_r(\mathbf{p})$ contains more finger points than palm points, then we put p in $\mathcal{P}_f$. Conversely, if the opposite holds, we put p in $\mathcal{P}_p$. The effect of this relabelling can be mitigated by moving a point from $\mathcal{P}_f$ to $\mathcal{P}_p$ only if half of hits neighbors, plus a slack, are in $\mathcal{P}_p$. The same can be done for the symmetric case. For most situations this refinement is able to adjust all wrong assignments.

Plane-based refinement Another refinement can be applied, consisting in moving in $\mathcal{P}_f$ all points whose distance from the hand plane is greater than a certain value. This approach can be particularly effective granted that a good quality estimation of the hand plane is available. The latter can be computed by running a RANSAC-like fitting algorithm on the set $\mathcal{P}_p$. Tests showed that for the critical cases the hand plane estimated this way is significantly more accurate than the one estimated using the same algorithm

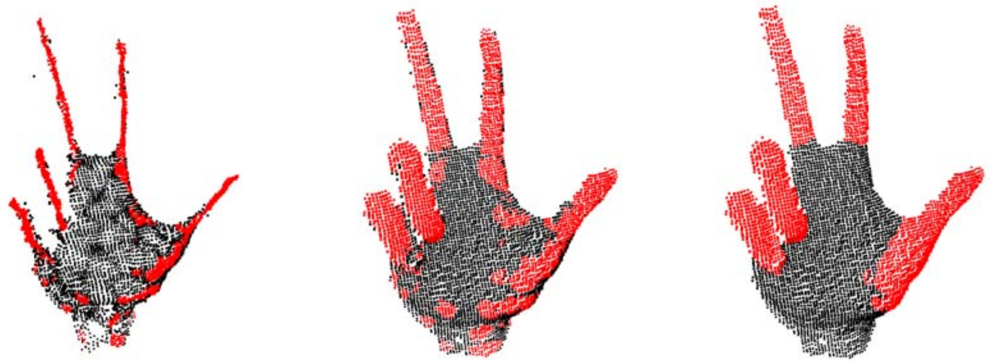

Figure 5.1: Palm-finger detection: in red points labelled as finger.

on $\mathcal{P}$.

Chapter 6

Finger Segmentation

6.1 Introduction

The previous chapter described how to achieve a first basic segmentation of the hand, by partitioning the point cloud into two subsets, $\mathcal{P}_f$ and $\mathcal{P}_p$, respectively the palm points and the finger points. In this chapter, this segmentation process is brought further, by trying to partition the set of finger points $\mathcal{P}_f$ into five subsets [1], each subset containing all and only those points belonging to a certain finger. More precisely, we want to partition the cloud so that if two points are both part of the same finger, then they included in the same partition and, vice versa if two points belongs to two different fingers, then they are included into different partitions. Unfortunately, as we will see, it will not be possible to achieve this goal, since the high variability in the hand cloud shape prevent the finding of an appropriate set of input parameters, for the clustering algorithm, that fits well for all situations. For the same reason, it is not a trivial task to elaborate some strategy that is able to automatically tune these parameters in order to get, each time, the desired solution.

The goal is to take points in $\mathcal{P}_f$ and separate them into five different subsets, each corresponding to a finger. In order to do so, the idea is to take again advantage of NGC transformation by attempting to separate the various fingers directly on the contracted finger point cloud. Indeed, another valuable property exhibited by NGC during tests was its ability to increase the separation between fingers, even if they are very close one another. The approach is to use a suitable clustering technique so that to end up with five clusters, one for each finger. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [24] was chosen to this purpose. This algorithm

[1]It is assumed that no finger is completely occluded.

has several advantages over other classical data clustering algorithms when applied to this kind scenarios, since it does not require to specify the number of clusters to search for, and it is able to discover noise and outliers. Furthermore, it is able to properly handle arbitrary shaped clusters. DBSCAN requires two parameters: the radius ε of the sphere to be used to check if neighbors are reachable, and the minimum number of points n_{min} that have to be in the ε-neighborhood to consider the point in the cluster. Since the high variability in shape and density of the clouds it has to deal, no values for these two parameters were found during tests that fits well for all situations. In particular, for any given values, there are cases when DBSCAN is able to discover exactly the five clusters corresponding to the five fingers, as well as cases when two or more fingers are clustered together or, conversely, the same finger is split into a number of smaller clusters. No automatic tuning strategy was found during the research in order to come up with this problem. Also, Ordering Points to Identify the Clustering Structure (OPTICS) [25, 26, 27] was considered, to use it as a tool for selecting the pair of values for (ε, n_{min}) which are more likely to lead DBSCAN in finding five clusters. However, results were not satisfactory. In the following two possible strategies are briefly illustrated, in which an approximate solution is allowed in exchange for a more predictable behaviour in the returned solutions.

6.2 Under Segmentation

This strategy accounts for solutions with less than five clusters. This means that at least one cluster contains points belonging to different fingers. Clearly, admitting this kind of solution requires a far less accuracy in the tuning of DBSCAN parameters than it would be if only solutions with (exactly) five clusters has to be produced. Moreover, results are more predictable, and it may be not hard to set (ε, n_{min}) so that, for most of cases, no finger is split in two different clusters. Having this guarantee, one possible approach is to under-segment $\mathcal{P}_f$ and then use some algorithm to split the clusters so that to properly come up with five finger clusters.

6.3 Over Segmentation

This strategy works in the opposite sense with respect to the strategy presented in the previous section. In particular, DBSCAN parameters are split so that more than five clusters are discovered, and at least one finger has been split into two or more small clusters. On the other side, it is true with high

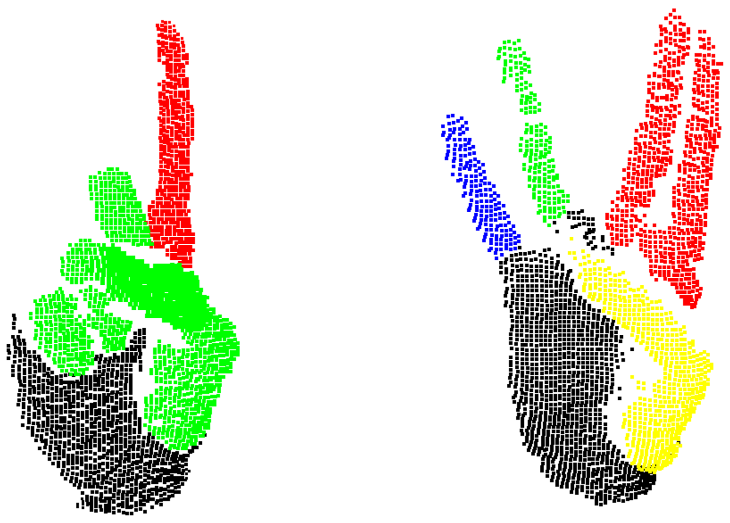
Figure 6.1: Under-segmentation using DBSCAN.

probability that no cluster is formed by merging points belonging to different fingers. This is the strategy implemented by the approach presented in this thesis. In particular, in the next chapter, a Linear Mixed Integer Program will be formulated in order to tackle this problem and appropriately group together clusters produced by a DBSCAN onver-segmentation.

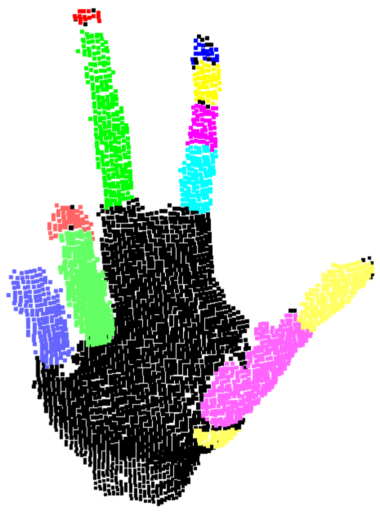
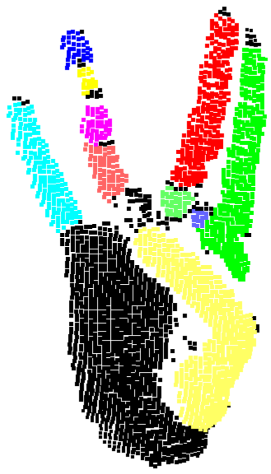

Figure 6.2: Over-segmentation using DBSCAN.

Chapter 7

A Linear MIP Formulation

In the previous chapter two strategies have been illustrated in order to come up with more predictable results when performing DBSCAN clustering, at the expense of a low-quality solution. In particular, this chapter concentrates on how to take advantage of an over-segmentation in order to achieve a correct classification of finger points. The approach is based on the mathematical programming paradigm, and passes through the formulation of a Mixed Integer Linear Program. This gives it the potential to be further improved, besides the present simple implementation, by enhancing the MIP formulation with new constraints and objectives, and by fully take advantage of the most efficient tools and strategies of mathematical optimization theory. At the same time, there are cases in which the generality of this approach can shows as a defect, especially if its performances are compared with other ad-hoc methods, and if no particular countermeasure is taken to better adapt its action to the specific case. The right way to use the MIP is in tandem with other procedures, or after some kind of data preprocessing to limit as much as possible its search space. Indeed, this is also the rationale followed here, since much of its potential strictly depends on the over-segmentation attained by DBSCAN.

Several other possibilities were considered, like using a mean-shift clustering [28] on a distance matrix computed by measuring the level of spatial alignment between each pair of clusters. The alignment was calculated by considering all the lines passing through two points, the first in one cluster and the second in the other cluster, and finding the line having the largest number of points at a small distance from it using a RANSAC approach. The ratio between the number of close points and the total number of points in the two clusters was taken as their alignment.

The idea is to take advantage of the over-segmentation returned by DBSCAN to constrain the projection of a hand skeleton model to the real hand cloud. In the present implementation a 21 joints model will be used, as the one depicted in the figure 7.1.

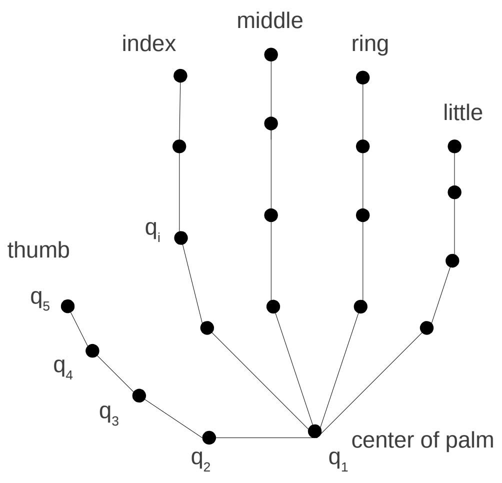

Figure 7.1: Hand skeleton model with 21 joints.

Let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be the hand PCD and let $\mathcal{Q} = \{q_1, \ldots, q_m\}$ be the set of joints in the model. Moreover let $\mathcal{F} = \{1, 2, 3, 4, 5\}$ be a set of five labels, each one corresponding to a specific finger. Finally, let $\mathcal{C} = \{c_1, \ldots, c_r\}$. The method aims in finding a good projection of the model onto the real hand point cloud. This is formulated as the problem of computing an assignment function a:

$$a : \mathcal{Q} \longrightarrow \mathcal{P}$$

with a series of additional constraints to be met in order for the assignment map to be feasible. As it will be clear in the next section, two sets of auxiliary variables, the slack variables and the cluster variables, will be necessary to add particular sets of constraints while maintaining linearity.

7.1 A Linear MIP Formulation

The program follows an Assignment Problem-like formulation, with additional constraints and variables.

Assignment variables

$$x_{i,j} = \begin{cases} 1 & \text{if} \quad a(q_i) = p_j \\ 0 & \text{otherwise} \end{cases} \qquad \forall 1 \leq i \leq n, \forall 1 \leq j \leq m$$

that is the variable $x_{i,j}$ is set to one if the joint q_i in the skeleton model is assigned to the point p_j in the point cloud $\mathcal{P}$; otherwise, the variable is set to zero.

Cluster variables

$$y_{f,k} = \begin{cases} 1 & \text{if} \quad f \mapsto c_k \\ 0 & \text{otherwise} \end{cases} \qquad \forall 1 \leq f \leq 5, \forall 1 \leq k \leq r$$

that is the variable $y_{f,k}$ is set to one if cluster c_k is labelled with label f, otherwise it is set to zero.

Slack variables

$$s_{i_1,i_2} \in \mathbb{R}_{\geq 0} \qquad \forall i_1, i_2 \quad \text{s.t.} \quad q_{i_1}, q_{i_2} \quad \text{adjacent joints}$$

They are continuous variables, and they are introduced in order to take into account for a dilation of convex polytopes forming the relative position constraints.

Assignment constraints

$$\sum_{1 \leq i \leq m} x_{i,j} \leq 1 \qquad \forall 1 \leq j \leq n \tag{7.1}$$

$$\sum_{1 \leq i \leq n} x_{i,j} = 1 \qquad \forall 1 \leq j \leq m \tag{7.2}$$

that is, each point in $\mathcal{P}$ is the target of at most one joint in $\mathcal{Q}$, while at the same time each joint in $\mathcal{Q}$ is assigned to some real point in $\mathcal{P}$.

Clustering constraints

$$\sum_{1 \leq f \leq 5} y_{f,k} \leq 1 \qquad \forall 1 \leq k \leq r \tag{7.3}$$

$$\sum_{i,j \in c_k} x_{i,j} - |c_k| \cdot y_{f,k} \leq 0 \qquad \forall 1 \leq f \leq 5, \forall 1 \leq c \leq r \tag{7.4}$$

where the summation is taken with i varying among all joints belonging to the finger with assigned label f. These constraints are quite cumbersome, yet their action is simple: prevent that two joints belonging to different fingers of the model are assigned to points in $\mathcal{P}$ that are part of the same cluster. These constraints are a direct application of the over-segmentation strategy: points in the contracted cloud $\mathcal{P}_H^c$ are clustered so that each cluster is completely included in a same finger, i.e. the cluster does not contain points from different fingers.

Relative position constraints Let q_{i_1} and q_{i_2} be two adjacent joints in the skeleton model. The following constraints basically constrain the relative position of the images of the two joints to be belong to some 3D convex polytope $\mathcal{R}_{i_1,i_2}$. By definition, $\mathcal{R}_{i_1,i_2}$ can be specified as the intersection of a finite numbers of half-spaces in $\mathcal{R}^3$. Let $b_1x + b_2y + b_3z + b_4 \leq 0$ be the inequality defining one of these half-spaces. Then, the following constraint must hold in order for the solution to be feasible:

$$b_1\Big(a(q_{i_1})_x - a(q_{i_2})_x\Big) + b_2\Big(a(q_{i_1})_y - a(q_{i_2})_y\Big) + b_3\Big(a(q_{i_1})_z - a(q_{i_2})_z\Big) + b_4 - s_{i_1,i_2} \leq 0 \tag{7.5}$$

where $a(q_i)_x$, $a(q_i)_y$, $a(q_i)_z$ are the x, y, z-coordinate of $a(q_i) \in \mathbb{R}^3$ respectively. As it can be seen, the slack variable $s_{i_1,i_2} \geq 0$ accounts for a dilation of the polytope.

Objective

$$\min \sum_{i_1,i_2} s_{i_1,i_2} \tag{7.6}$$

Chapter 8

Results

In order to evaluate the performance of the proposed approach, we acquired a dataset of gestures using a Creative SENZ3D depth camera. The dataset has been acquired in our laboratory and contains 20 different gestures repeated 6 times for a total of 120 different acquisitions. Color data has also been acquired but notice that it is not used in the proposed approach. A sample image and depth map for each of the gestures is shown in Fig. 8.1. We used Meshlab in order to get access to a direct and friendly visualization of all intermediate and final results at each step of our work. We store each PCD we deal with in the Stanford PLY format, which gives the possibility to specify 3D points, edges and facets in a relatively simple way, as a list where each row corresponds to a single occurrence of one of these three types of objects. Moreover, it also give the possibility to indicate, for each object, the RGB color in which it should be displayed. This feature revealed particularly suitable to visually represent the labels assigned to the points in the PCD by the clustering algorithms we used, as well as to easily evaluate the quality of finger and palm points extraction and various other types of partitioning. The latest version we used, Meshlab 1.3.3, can be directly downloaded from `http://meshlab.sourceforge.net`. Finally, to solve the Linear MIP presented in Chapter 7 we used IBM Ilog Cplex [29].

Fig. 8.2 shows some sample depth maps and the corresponding outputs of the proposed algorithm, i.e. palm detection and fingers classification. As expected, the proposed algorithm correctly handles the simplest situations, e.g., the open hand shown in the first column. The second and third columns show how fingers bent over the palm are correctly identified, a quite critical scenario for several approaches, specially if based on silhouette or shape analysis. Note how the complex configurations of the last two columns are also handled correctly by our approach, fingers are very close to each other and there are significantly occluded regions. In all these examples then, the

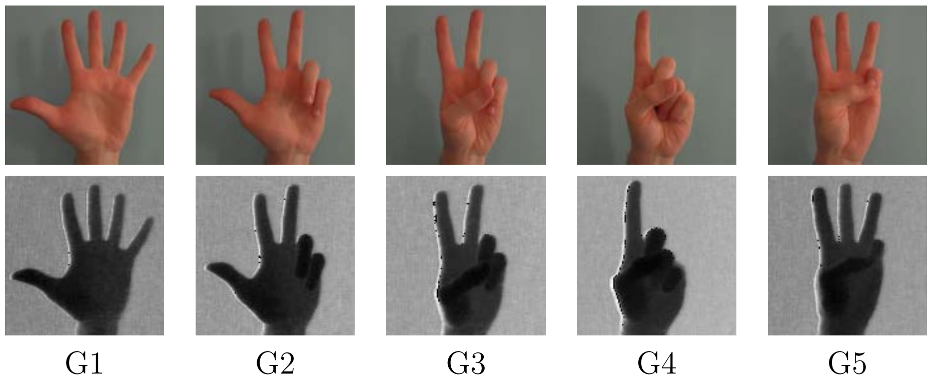

Figure 8.1: Sample images and depth maps for each of the gestures in the database.

palm region is well detected as can be noticed from the second row of Fig. 8.2.

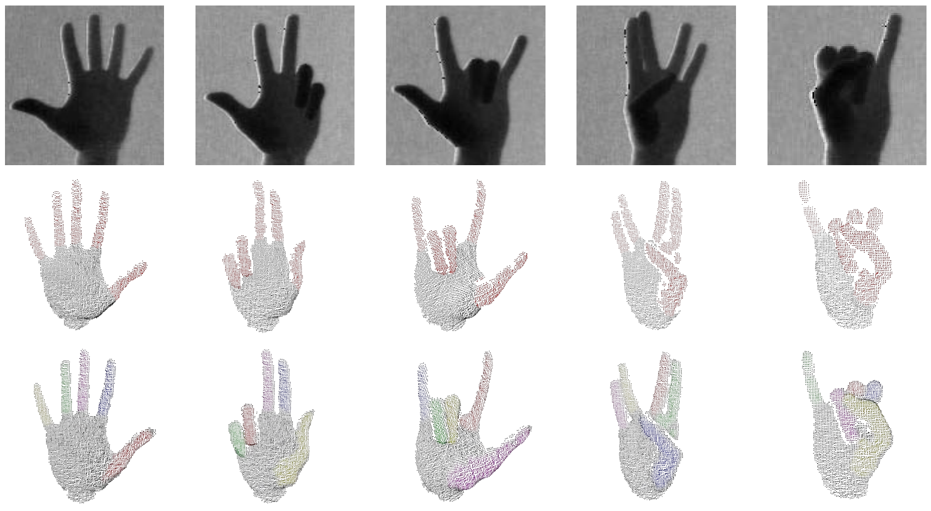

Figure 8.2: Output of the proposed approach on some sample gestures: (first row) Input data; (second row) Palm and fingers regions; (third row) Fingers classification. (*Best viewed in colors*).

The scheme for palm detection is very accurate, the correctness has been evaluated by means of visual inspection and in 97% of the gestures the palm region was entirely correctly assigned. In the few remaining gestures, some isolated spots have been assigned to fingers in the region of the palm. Table 8.1 shows in the first column the average accuracy of the proposed approach for fingers classification on the considered dataset. As expected, simpler gestures are very well identified (e.g., gestures G1, G2 and G6), and even gestures where fingers are very close to each other (e.g., G15, G16 and G19) of bent over the palm (e.g., G3, G8, G9) result in a very good accuracy. The dataset contains a limited but challenging number of gestures, therefore a little variation in the correctness of the recognition process leads to large variation in the results. However, differently from other datasets, that contain a continuous acquisition producing many frames, we collected a set of gestures very different from each other.

The other three columns report the kind of error that leads to the wrong identification. On average, only in the 1.7% of the acquisitions the algorithm is not able to detect all the 5 fingers, while 20% of the times the fingers classification step collapse more than one finger together. This is seldom due to a bad classification after the linear assignment problem, indeed, only 7.5% of the time it classifies one finger into two separate clusters, most of the time instead, there are some residual spots around a finger cluster that lead to ill-conditioning the problem. The worst classified gestures are G4 and G17, in the first there are too many occlusions, while in the second, the acquired depth map does not present sufficient surface details to discriminate fingers from a planar region. In almost all the failure cases however, two adjacent fingers are collapsed together, and the overall fingers classification without considering this region can be retained correct.

Gesture	Correctly recognized	Missing Fingers	Joined Finger	Splitted Fingers
G1	100%	0%	0 %	0%
G2	100%	0%	0 %	0%
G3	83.3%	0%	16.7 %	16.7%
G4	16.7%	0%	83.3 %	0%
G5	100%	0%	0 %	0%
G6	100%	0%	0 %	0%
G7	66.7%	16.7%	16.7 %	0%
G8	100%	0%	0 %	0%
G9	100%	0%	0 %	0%
G10	83.3%	0%	16.7 %	0%
G11	66.7%	16.7%	16.7 %	0%
G12	83.3%	0%	16.7 %	0%
G13	66.7%	0%	33.3 %	0%
G14	50%	0%	50 %	50%
G15	100%	0%	0 %	0%
G16	83.3%	0%	16.7 %	0%
G17	33.3%	0%	66.7 %	33.3%
G18	66.7%	0%	16.7 %	33.3%
G19	83.3%	0%	16.7 %	0%
G20	50%	0%	50 %	16.7%
AVERAGE	76.7%	1.7%	20.8 %	7.5%

Table 8.1: Accuracy of the proposed approach on our database. The first column shows the accuracy of the proposed approach (i.e., the percentage of correctly assignments). In case of wrong recognition the other three columns show the type of error made by the algorithm, i.e., if a finger is not detected or is splitted in two or more parts and if multiple fingers are joined together. Notice that since multiple errors can be made on the same acquisition in some cases the sum can be greater than 100%.

Chapter 9

Conclusions

In this thesis an efficient approach was proposed for the recognition of the palm and fingers from a single depth map without exploiting temporal constraints. The palm and fingers regions are discriminated by contracting the 3D point cloud along the normal directions and analyzing the point density in the output of this process. This allows to recognize also fingers bent over the palm, a quite critical issue for many silhouette and shape-based approaches. Density-based clustering is then used to perform an over-segmentation of the fingers regions and finally the segments are associated to the various fingers by an integer linear programming approach. Experimental results demonstrate the effectiveness of the approach on challenging datasets containing complex gestures with inter-occlusions and fingers bent over the palm and over other fingers. Notice how after recognizing the various fingers it is much simpler to recognize the hand pose by associating the fingers depth samples to the hand skeleton, further research will be devoted to the exploitation of the proposed approach for hand pose estimation.

Appendices

Appendix A

Principal Component Analysis

Principal Component Analysis, shortly referred to as PCA, is a powerful yet easy-to-implement method which is able to decorrelate a set of two or more variables through the application of a particular linear map. More precisely, it allows the mapping of a set of possibly correlated variables into a new set of uncorrelated variables, sorted in order of decreasing variance, the first of them taking into account for the most variation present in the original set. These new variables are also called Principal Components.

PCA is one of the simplest and most used tools of multivariate analysis, and it finds direct application in many fields such as computer vision, where its geometrical interpretation is exploited in order to find best-fitting planes and lines, or data compression [30], where it is used as a basic step to perform transform coding. PCA approach was introduced for the first time by K. Pearson in 1901 [31], although a precise formalization was provided by H. Hotelling in 1933[32].

In order to illustrate the machinery behind PCA - with no claim of being exhaustive - we briefly discuss how the first principal component can be derived, following the statistical approach presented in [33]. We start considering a set of n real variables $x = [x_1, ..., x_n]^T$, whose covariance matrix Σ is:

$$\Sigma = E[(x-\mu)(x-\mu)^T] = \begin{bmatrix} \sigma_1 & \sigma_{1,2} & \cdots & \sigma_{1,n} \\ \sigma_{2,1} & \sigma_2 & \cdots & \sigma_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ \sigma_{n,1} & \sigma_{n,2} & \cdots & \sigma_n \end{bmatrix} \tag{A.1}$$

where $\mu = E[x]$ is the mean expectation of x, $\sigma_{i,j} = E[(x_i - \mu_i)(x_j - \mu_j)]$ is the covariance associated to each pair of variables $(x_i, x_j)_{i \neq j}$, while $\sigma_i = E(x_i - \mu_i)(x_i - \mu_i)^T]$ is the variance of each variable x_i. Note that, in the practical case, we have no access to the covariance matrix: however, we can compute a sample covariance matrix starting from a set of observations or samples of our variables, and successfully apply the method without major changes. Let M be the $m \times n$ sample matrix collecting m observations of the n variables, and let $c \in \mathbb{R}^n$ be the centroid of the samples, then we can build the right covariance matrix as $S = (M - c)^T (M - c)$. In order not to burden our notation, in what follows we shall assume x having zero mean, so that its covariance matrix can be simply written as $\Sigma = E[xx^T]$. In the real case, where we consider the sample covariance matrix, we shall perform all computations after having recentred all samples toward their centroid.

Our goal is to create a new variable z_1 which is a linear combination of the original set of variables and whose variance is the maximum among the variances of all variables that can be built out of a linear combination of x. That is, we search for a coefficient vector $a_1 \in \mathbb{R}^n$ such that $z_1 = a_1^T x$ has maximum variance $\text{var}[z_1] = \max_{a \in \mathbb{R}^n} \{\text{var}[z] : z = a^T x\}$. In order to avoid trivial solutions, we also constrain a_1 to have unit length, that is $a_1^T a_1 = 1$. The new variable z_1 will be referred to as the first principal component.

The problem, which is a maximization problem with one single constraint, can be concisely stated as follows:

$$\begin{array}{rcll} a_1 & = & \arg\max & a^T \Sigma a \\ & & \text{s.t.} & a^T a = 1 \\ & & & a \in \mathbb{R}^n \end{array} \tag{A.2}$$

where we have used the fact that $\text{var}[a^T x] = a^T x x^T a = a^T \Sigma a$, which in turn follows from our assumption on x having zero mean. We can then compute the first principal component as $z_1 = a_1^T x$. The problem can be solved using the Lagrangian relaxation, by removing the constraint cit and by including it in the objective function, weighted by some multiplier $\lambda \in \mathbb{R}$. In this way, we obtain the following relaxed problem:

$$\begin{array}{rcll} a_r, \lambda_r & = & \arg\max & a^T \Sigma a - \lambda(a^T a - 1) \\ & & & a \in \mathbb{R}^n, \lambda \in \mathbb{R} \end{array} \tag{A.3}$$

Differentiating the Lagrange function $L(a, \lambda) = a^T \Sigma a - \lambda(a^T a - 1)$ with respect to a gives:

$$\Sigma a - \lambda a = 0 \tag{A.4}$$

which says that if (a_r, λ_r) is an optimal solution to the relaxed problem, then λ_r is an eigenvalue of Σ and a_r is the corresponding eigenvector. Moreover, it can be proved that all eigenvectors of Σ are orthonormal vectors, since Σ is a positive semi-definite matrix. This implies that $a_r^T a_r = 1$, so that an optimal solution to the relaxed problem is also an optimal solution to the original problem. Finally, observe that:

$$L(a_r, \lambda_r) = a_r^T \Sigma a_r - \lambda_r (a_r^T a - 1) = a_r^T \Sigma a_r = a_r^T \lambda_r a_r = \lambda_r a_r^T a_r = \lambda_r \quad \text{(A.5)}$$

Thus, an optimal solution is given by taking λ_r as the maximum eigenvalue of Σ and a_r as its corresponding eigenvector. Going back to the original problem, we have that the first principal component for a set of variables x can be computed as $z_1 = a_1^T x$ choosing a_1 as the eigenvector of Σ with largest eigenvalue. In cit the approach to derive the first principal component is rather similar, except that it focus on minimizing the distances from the samples to a best-fitting hyperplane. We shall not show how to derive the 2-nd up to the n-th principal components: however, it can be proved in a similar manner that the k-th principal component is $z_k = a_k^T x$, where a_k is the eigenvector of Σ with the k-th largest eigenvalue.

Acknowledgments

Ringrazio il Professor Pietro Zanuttigh, per l'opportunità datami e per il supporto fornitomi lungo il corso di questo intenso ma entusiasmante periodo di ricerca. Ringrazio Giulio Marin, instancabile ricercatore ed inesauribile fonte di consigli e suggerimenti. Voglio ricordare Marta, per aver sopportato il mio forzato isolamento lungo il corso di questo difficile viaggio. Il ringraziamento più grande va ai miei genitori, che in questi anni mi hanno supportato con la loro pazienza. Infine, voglio ringraziare tutti i miei amici e colleghi universitari per i momenti felici trascorsi assieme.

Bibliography

[1] Rehg, J.M., Kanade, T.: Visual tracking of high dof articulated structures: an application to human hand tracking. In: Computer VisionECCV'94. Springer (1994) 35–46

[2] Erol, A., Bebis, G., Nicolescu, M., Boyle, R.D., Twombly, X.: Vision-based hand pose estimation: A review. Computer Vision and Image Understanding **108**(1) (2007) 52–73

[3] Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., Moore, R.: Real-time human pose recognition in parts from single depth images. Communications of the ACM **56**(1) (2013) 116–124

[4] Keskin, C., Kıraç, F., Kara, Y.E., Akarun, L.: Real time hand pose estimation using depth sensors. In: Consumer Depth Cameras for Computer Vision. Springer (2013) 119–137

[5] Xu, C., Cheng, L.: Efficient hand pose estimation from a single depth image. In: Computer Vision (ICCV), 2013 IEEE International Conference on, IEEE (2013) 3456–3462

[6] Oikonomidis, I., Kyriazis, N., Argyros, A.A.: Efficient model-based 3d tracking of hand articulations using kinect. In: BMVC. Volume 1. (2011) 3

[7] Qian, C., Sun, X., Wei, Y., Tang, X., Sun, J.: Realtime and robust hand tracking from depth

[8] Suryanarayan, P., Subramanian, A., Mandalapu, D.: Dynamic hand pose recognition using depth data. In: Pattern Recognition (ICPR), 2010 20th International Conference on, IEEE (2010) 3105–3108

[9] Ren, Z., Yuan, J., Zhang, Z.: Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In:

Proceedings of the 19th ACM international conference on Multimedia, ACM (2011) 1093–1096

[10] Dominio, F., Donadeo, M., Zanuttigh, P.: Combining multiple depth-based descriptors for hand gesture recognition. Pattern Recognition Letters (2013)

[11] Suau, X., Alcoverro, M., López-Méndez, A., Ruiz-Hidalgo, J., Casas, J.R.: Real-time fingertip localization conditioned on hand gesture classification. Image and Vision Computing (2014)

[12] `http://www.leapmotion.com/`

[13] Huhle, B., Schairer, T., Jenke, P., Straßer, W.: Robust non-local denoising of colored depth data. In: Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on, IEEE (2008) 1–7

[14] `http://us.creative.com/p/web-cameras/creative-senz3d`

[15] `http://www.microsoft.com/en-us/kinectforwindows/`

[16] `http://www.softkinetic.com/`

[17] `http://software.intel.com/en-us/vcsource/tools/perceptual-computing-sdk/home` (2013)

[18] Dominio, F., Donadeo, M., Marin, G., Zanuttigh, P., Cortelazzo, G.M.: 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, ACM (2013) 9–16

[19] Van den Bergh, M., Van Gool, L.: Combining rgb and tof cameras for real-time 3d hand gesture interaction. In: Applications of Computer Vision (WACV), 2011 IEEE Workshop on, IEEE (2011) 66–72

[20] `http://pointclouds.org/`

[21] Rusu, R.B., Marton, Z.C., Blodow, N., Dolha, M., Beetz, M.: Towards 3d point cloud based object maps for household environments. Robotics and Autonomous Systems **56**(11) (2008) 927–941

[22] Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: Computer Vision, 1998. Sixth International Conference on, IEEE (1998) 839–846

[23] Rusu, R.B.: Semantic 3d object maps for everyday manipulation in human living environments. KI-Künstliche Intelligenz **24**(4) (2010) 345–348

[24] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd. Volume 96. (1996) 226–231

[25] Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. In: ACM Sigmod Record. Volume 28., ACM (1999) 49–60

[26] Daszykowski, M., Walczak, B., Massart, D.: Looking for natural patterns in data: Part 1. density-based approach. Chemometrics and Intelligent Laboratory Systems **56**(2) (2001) 83–92

[27] Daszykowski, M., Walczak, B., Massart, D.L.: Looking for natural patterns in analytical data. 2. tracing local density with optics. Journal of chemical information and computer sciences **42**(3) (2002) 500–507

[28] Cheng, Y.: Mean shift, mode seeking, and clustering. Pattern Analysis and Machine Intelligence, IEEE Transactions on **17**(8) (1995) 790–799

[29] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

[30] Sayood, K.: Introduction to data compression. Newnes (2012)

[31] Pearson, K.: On lines and planes of closest fit to systems of points in space. Philosophical Magazine (2) (1901) 559–572

[32] Hotelling, H.: Analysis of a complex of statistical variables into principal components. Educational Psychology (24) (1933)

[33] Jolliffe, I.: Principal component analysis. Wiley Online Library (2005)