VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SYSTEM FOR RECOGNITION OF 3D HAND GEOMETRY

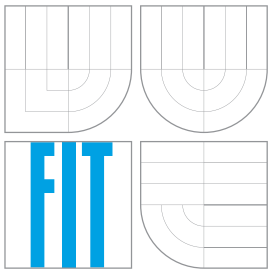
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN SVOBODA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SYSTÉM PRO ROZPOZNÁVÁNÍ PODLE 3D GEOMETRIE RUKY

SYSTEM FOR RECOGNITION OF 3D HAND GEOMETRY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAN SVOBODA
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. MARTIN DRAHANSKÝ, Ph.D.
SUPERVISOR

BRNO 2014

Abstract

In the last decade, there has been an increased interest in using 3D data for biometric person recognition. Perhaps the most widely researched application is 3D face recognition, where several commercial products are currently available on the market. There have been some research works on the 3D hand recognition as well, however, no commercially viable systems are currently known. Independently, in the recent years inexpensive 3D sensors have become a commodity, potentially enabling a wide range of 3D biometric applications. The main goal of this work is to develop a functioning prototype of a touchless 3D hand recognition system based on a new cheap RealSense 3D camera developed by Intel. One of the challenges in using the RealSense camera is that due to this small form factor, it produces relatively low quality samples in comparison to the more expensive acquisition hardware used in the previous research on the 3D hand biometrics. We analyze the robustness of different 2D and 3D features and study several methods for their fusion. We evaluate the performance of the system, showing that it achieves results comparable with the state-of-the-art.

Abstrakt

V posledním desetiletí došlo ke zvýšení zájmu o užití 3D dat k biometrické identifikaci osob. Možná vůbec největší výzkum proběhl v oblasti 3D rozpoznávání podle obličeje, přičemž je v současné době dostupných vícero komerčních zařízení. V oblastni rozpoznávání podle 3D geometrie ruky byl v minulých letech proveden určitý výzkum jehož výsledkem však nebylo žádné komerční zařízení. Nezávisle na tomto výzkumu se v posledních letech velmi rozšířil trh s cenově dostupnými 3D sensory, což potenciálně umožňuje jejich nasazení v mnoha typech biometrických systémů. Hlavním cílem této práce je vytvořit funkční vzorek bezdotykového systému pro rozpoznávání osob podle 3D geometrie ruky, který bude používat novou levnou kameru RealSense 3D vyvíjenou v současné době firmou Intel. Jedním z problémů při použití RealSense kamery je její velmi malý form factor, který je příčinou nižší kvality výsledných snímků v porovnání s velmi drahými alternativami, které byly použity v již dříve zmíněném výzkumu 3D biometrických systémů. Práce se snaží analyzovat robustnost různých 2D a 3D příznaků a vyzkoušet několik různých přístupů k jejich fúzi. Rovněž je vyhodnocena výkonnost výsledného systému, kde je ukázáno, že navržené řešení dosahuje výsledků porovnatelných se state-of-the-art.

Keywords

surface reconstruction, 3D reconstruction, computer vision, biometric systems, biometry, 3D hand, hand biometrics

Klíčová slova

rekonstrukce povrchu, 3D rekonstrukce, počítačové vidění, biometrické systémy, biometrie, 3D ruka, biometrie ruky

Citace

Jan Svoboda: System for Recognition of 3D Hand
Geometry, diplomová práce, Brno, FIT VUT v Brně, 2014

System for Recognition of 3D Hand Geometry

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Martina Drahanského.

.....
Jan Svoboda
July 29, 2014

Poděkování

Rád bych poděkoval Prof. Michaelu Bronsteinovi a doc. Martinu Drahanskému za odborné vedení mé diplomové práce a mnoho cenných návrhů v průběhu řešení. Velký dík patří rovněž výzkumné skupině STRaDe z FIT VUT v Brně a také společnosti Touchless Biometric Systems s.r.o. za poskytnuní podpory a zdrojů. V neposlední řadě musím poděkovat rovněž těm z mých přátel, kteří mi pomohli s revizí anglické gramatiky.

Contents

Chapter 1

Introduction

There has been a significant amount of research in the field of biometric systems in the past few years. It has brought us new technologies and possibilities considering security systems in general. Many parts of the human body have been studied as biometric characteristics and there has been considerable effort towards implementing various security systems based on those characteristics. However, we are still far from being able to say that all the potentially good biometric characteristics can be employed for building a commercially viable biometric system. Nowadays, the commercial biometric systems use mainly three characteristics: fingerprints, face and hand.

This work focuses on hand biometric systems. Talking about hand based biometric systems, there are working, commercially used solutions based on the 2D hand geometry. Considering the 3D hand geometry, there have been several attempts to use the 3D information in the academic research community, with promising results. Besides improved accuracy and robustness to hand pose, one practically appealing aspect of the 3D hand recognition is a touchless scenario (as opposed to standard 2D systems in which the hand is typically placed on a reflecting background). However, most of the academic works on 3D hand recognition are based on high-end acquisition devices that on the one hand provide very high (sub-millimeter) resolution, but they are too cumbersome and expensive to be part of a viable commercial system on the other.

The main focus of this work is to develop a prototype of 3D hand recognition system based on such an inexpensive sensor.

The rest of the thesis is organized as follows. In Chapter 2, we review some basic notions in biometric recognition systems and current state of the art and practice. Next, in Chapter 3, we describe the proposed solution. Evaluation of the created system is documented in Chapter 4, providing information about how well was the goal achieved. The whole text is enclosed with a short summary, discussing several future research directions and improvements as well.

Chapter 2

Hand based biometric systems

The goal of this chapter is to provide the reader with a brief introduction into the biometric systems based on the human hand properties. Presented knowledge is the basis for understanding the rest of this text.

In the first section, methods for acquisition of 3D models are explained. Also, acquisition devices that are currently available on the market are shortly described. Then, in the Section 2.2, focus is shifted towards the recognition of people according to their hand geometry. It summarizes the research that has been already done in this field. Next, Section 2.3 explains the problematics of multimodal biometric systems. In Section 2.4, methods for the evaluation of the biometric systems are presented. At the end, Section 2.5 is dedicated to the existing industrial systems for hand geometry based recognition of people.

2.1 Acquisition devices

The selection of the acquisition device (sensor) is one of the key decisions in the design of a biometric system. On on hand, the form factor and cost of the sensor are important, as they have a direct impact on the packaging and the bill of materials (BOM) of the final product. On the other hand, the data acquired by the sensor must be of high quality, allowing to reduce or avoid the post-processing of the acquired data. Usually, those two criteria are in conflict (a higher quality sensor is larger and more expensive and vice versa) and one has to carefully select a good trade-off.

The acquisition devices for 2D hand recognition systems are usually a simple cameras (typically working in the visible, UV or IR range), mounted above (for system based on the dorsal side of the hand) or below (for hand palm side) the area where the hand is placed.

Speaking about 3D case, there are a lot of possibilities, such as passive stereo analysis, photometric stereo, structured light approach and many others [11]. In the following section, a few popular approaches for the 3D data acquisition are described.

2.1.1 Available methods for 3D data acquisition

Nowadays, plenty of methods for 3D reconstruction are already known and there is still a big ongoing research in this field. When choosing a proper method, not only its precision has to be taken into account. There are many other aspects, such as price of the components, constraints that each method brings, etc.

Passive stereo

As stated in [11], passive stereo analysis is based on principles very close to the human vision. The human vision is fundamentally a binocular process that takes two images obtained from a slightly different viewpoints and estimates the depth from the parallax between the two images. Big advantage of the passive stereo methods is that no other devices are needed for the reconstruction process, no special light sources, no pattern projectors, etc.

Passive stereo devices use that at least two cameras are capturing the scene at the same time, or within a certain interval during which no objects are moving in the scene. The whole reconstruction process is based on principles of so-called epipolar geometry, which is the expression for the geometry of the stereo vision and is explained for example in [11]. As also described in [11], passive stereo pipeline usually consists of the following stages:

- **Image acquisition** - acquisition of the images that will be further processed;
- **Camera modeling** - represents the acquisition device (mainly camera) calibration;
- **Feature extraction** - detection of the image significant features that are needed later on;
- **Correspondence analysis** - matching of the corresponding image points in images from both cameras;
- **Triangulation** - position of a certain point in the 3D space can be calculated given two corresponding points, one from each camera. For detailed description of the triangulation please refer to [6];
- **Interpolation** - original calculated 3D points often have to be transformed into better representation of the object surface.

Examples of stereo-based 3D reconstruction are shown in Figures 2.1(a), 2.1(b) and 2.1(c).For additional details, please refer to Chapter 4 of [11].

(a) (b) (c)

Figure 2.1: (a) Input images from both cameras; (b) Disparity map between the images; (c) 3D model of the scene reconstructed using the disparity map [4].

Photometric stereo

This method is based on SFS (Shape From Shading) methods, which are explained in detail in [11]. It extends the SFS methods by using not only one image of the scene, but more of them. Classic SFS methods recover the object surface from a single irradiance image using reflection properties of the object and illumination parameters in the scene. The photometric stereo uses multiple light sources to obtain several views of the scene with

a different illuminations (unlike stereo-based approaches, at least three independent light directions are needed). The different illuminations can be captured at different times or together using color filters. The depth reconstruction is performed by first recovering the surface orientations from the illumination images and then integrating them into a surface.

An example of the photometric stereo processing pipeline is in Figure 2.2. For more details about these methods, refer to chapter 8 of the book [11].

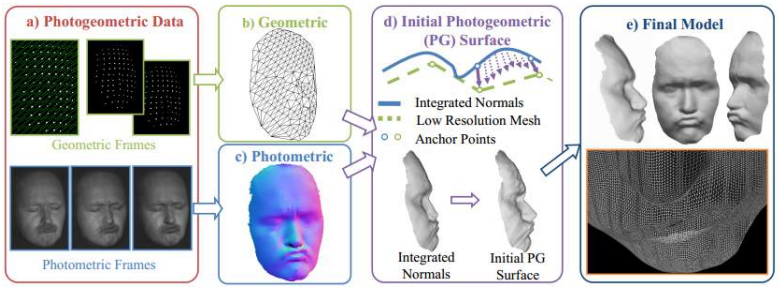
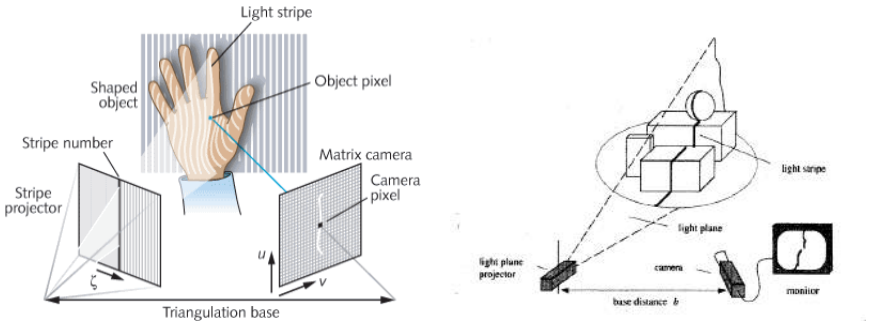


Figure 2.2: Processing pipeline of the photometric stereo approach using three light sources. Here the photometric stereo approach is combined with previously obtained sparse geometric data [22].

Structured light

In these systems, the camera is viewing the scene at a certain angle with respect to the light pattern source (projector), which is designed to project special patterns onto the scene. Thus the light pattern projected onto the object is distorted while being viewed from the camera point of view. Knowing the parameters of the camera - light source system, the position of the object surface 3D points can be computed.

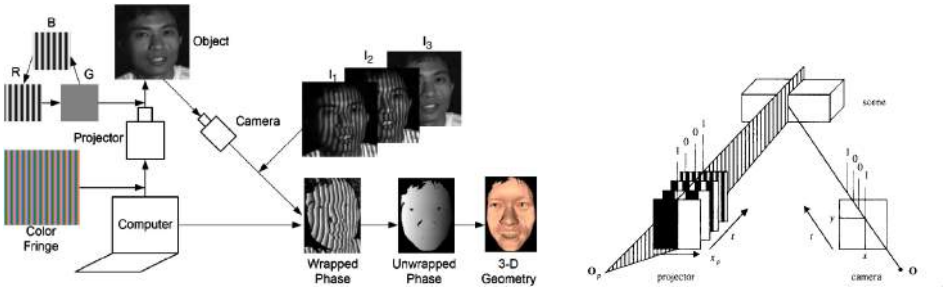


(a) (b)

Figure 2.3: (a) Multiple parallel stripes pattern; (b) Single laser stripe pattern [11].

As stated in [11], structured light approach can be regarded as a modification of static binocular stereo. Only in this case, one of the cameras is replaced by a light source that projects the light pattern into the scene. The triangulation is then carried out by intersecting the projection ray that is casted from the camera into the scene and the light ray (or plane) which is going into the scene from the light source.

There are many light patterns that can be used. Basically, methods can be divided into two groups. First group uses simple geometric patterns, such as dot patterns (single or multiple dots), stripe patterns (again single or multiple lines), etc., see examples in Figure 2.3. The second group includes methods that are based on a spatial or temporal coding of the light patterns, such as binary encoded light stripes approach or phase shifting, as shown in Figure 2.4.



(a) (b)

Figure 2.4: (a) Two plus one phase shifting method processing pipeline;[23] (b) Binary encoded patterns method [11].

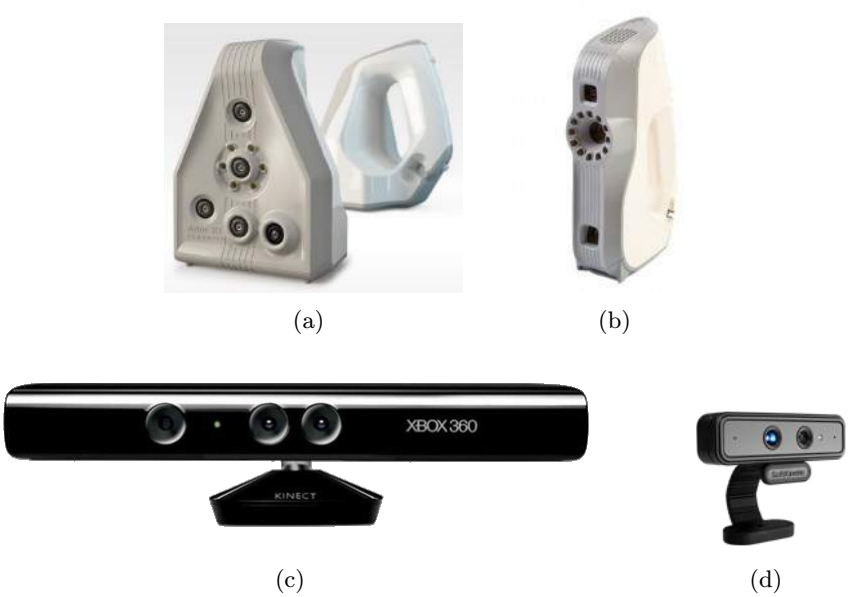
Structured light methods are of high precision and they simplify the whole reconstruction process from the mathematical (theoretical) point of view by making the hardware acquisition system more complex.

2.1.2 Existing devices

A variety of 3D sensors is currently available on the market. They differ not only in precision and price, but also in the principles they are using for the 3D surface reconstruction, the form factor, temporal, spatial, and depth resolution. Notable manufacturers include big companies such as Microsoft as well as more of a start-up companies such as Luxembourg-based Artec 3D, Israeli PrimeSense (recently acquired by Apple) and Belgian SoftKinetic.

Considering the high-end very precise sensors, already mentioned **Artec 3D** (Figure 2.5) sensors are perfect examples. These sensors are very precise and they can satisfy very demanding customers, on the other hand the price is really high, in range of tens of thousands of euros, which makes them almost useless in case of industrial biometric systems. In case of biometric systems, the focus is shifted towards cheaper, yet still precise enough sensors.

In the lower-end range one can find sensors that cost a few hundreds of euros at most. **Microsoft Kinect** (Figure 2.5(c)) was the first device of its kind that provided cheap 3D sensor, which, apart from augmented reality use, could also serve as a 3D scanner. After Kinect was released, many other sensors based on similar approach were developed and sold by companies like **SoftKinetic** (Figure 2.5), etc. Finally, there are on-going attempts to reduce the price and dimensions of the sensors even further. A recent example includes a 3D camera announced by Intel (Figure 2.6), which was used for this work.



(a) (b) (c) (d)

Figure 2.5: Artec3D handheld sensors: (a) Artec 3D Spider; (b) Artec 3D Eva. Cheap 3D sensors: (c) Microsoft Kinect; (d) SoftKinetic DS325.

2.2 Human hand as biometric characteristic

Far back in the second half of the 19th century, Alphonse Bertillon developed anthropometry, which describes what measurements of the human body have to be done in order to identify individuals. These measurements include also three values obtained from the hand contour geometry. Since then, hand geometry was used for identification of people.

The human hand provides sufficient amount of information for performing both 2D and 3D recognition. 2D recognition is done based on the hand contour, where the precise extraction of the contour is the most critical part. Considering 3D hand geometry, both hand palm (and in particular, the structure of the wrinkles and epidermal ridges of the so-called palmprint) and hand dorsal side can be used in order to extract biometric features for the recognition process. In case of the hand palm, not only the 3D structure is available, but so-called palmprint could be also used to get even more information from the hand surface.

Biometric systems based on the 2D hand geometry are quite widespread nowadays. The overall experience from the field is that users do not complain about having to use such systems. However, there is one thing that is not accepted very well by users. It is the need to place the hand somewhere in order to be recognized.

The biggest advantages of the hand geometry properties are:

- good user acceptance;
- easy to use;
- robust in many environments.

On the other hand, there are also a few disadvantages that have to be taken into account:

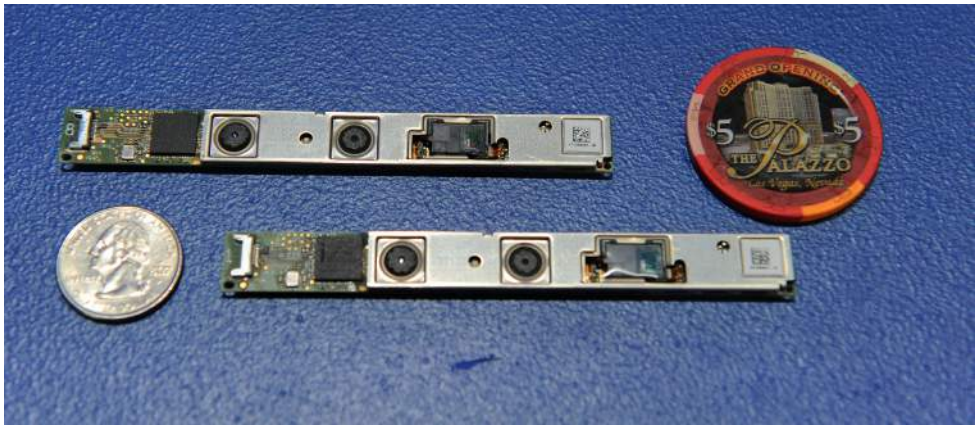


Figure 2.6: The Intel RealSense camera.

- bigger acquisition devices in comparison with some other approaches (e.g. fingerprints, etc.);
- using only 2D hand geometry does not provide enough distinctive power for larger groups of people;
- users may complain about having to place their hand on a particular surface.

Much of exploration of the 3D hand recognition systems has been motivated by the two latter limitations. The use of additional 3D geometric structure of the hand, in combination with already existing 2D hand geometry, can improve the recognition accuracy, and also eliminate the need of hand placement on a particular surface, making the system touchless.

Recognition is based on the assumption that human hand is unique. As listed in [2], to recognize individuals, the following features are currently used:

- finger length;
- finger width;
- finger height;
- curvatures and local anomalies.

2.2.1 2D geometry

As stated in [2], there are three main approaches to recognition based on the 2D hand geometry. They are based either on direct measurements, hand alignment or analysis of finger widths.

Direct measurements

In this approach, some significant proportions of the human hand are measured. Based on those proportions, the actual recognition is done afterwards. During the image acquisition, hand has to be placed on a surface that sometimes also has a few guide pins that help to position the hand and the fingers properly.

Typically, the measured proportions are finger lengths, finger widths at different parts of the finger, palm width, etc. as shown in Figure 2.7. These proportions can be extracted from the hand image captured by the camera (due to the use of reflective surface, the hand is clearly segmented from the background) and constitute the feature vector that characterizes a hand.

Recognition is done taking two templates that contain the feature vectors and computing a distance measure between those two templates. As shown in [2], given two feature vectors x and y, the comparison score is computed using some standard distance, such as L1 or L2.

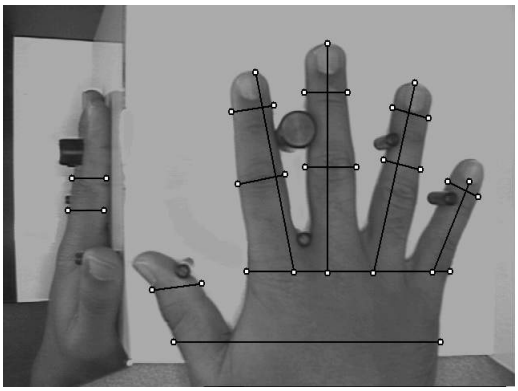


Figure 2.7: Proportions of the hand measured in the direct measurements approach [2].

Hand alignment

Hand alignment is essentially template matching. First, each captured image is aligned into predefined position and then the original template and then compared to the template image. For each alignment, the so-called mean alignment error (MAE) is computed, which is the average distance between the corresponding points. If the MAE value is below specified threshold, hands are considered as similar. The whole process is illustrated in Figure 2.8. This method is usually used only for verification.

Approach based on analysis of finger widths

First of all, Otsu's algorithm [2] is used to separate the hand region from the background. Next, axes of the hand are estimated, as shown on Figure 2.9(c). Using these axes, positions of the finger tips and finger valleys are found, see Figure 2.9(d). Using this information, hand blob is divided into separate fingers. Now, as the fingers are separated, the distance from each finger edge point to the finger center axis is computed. This is visualized in Figure 2.9(f). For each finger, from all the distances computed on it, histogram that represents the probability distribution of the distances is created. The feature vector describing the hand comprises these histograms.

During comparison the divergence between the histograms representing each finger is computed using Kullback-Leibler [1] distance. The final distance is the sum of the three smallest distances. More detailed description of this method can be found again in [2].

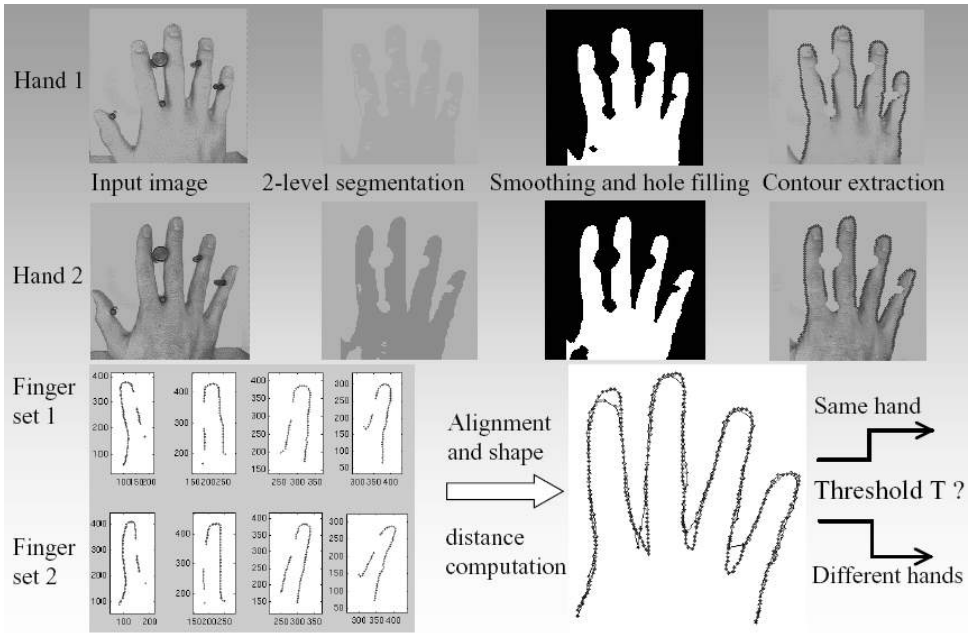


Figure 2.8: Processing pipeline of the hand alignment method [2].

2.2.2 3D geometry

To best of our knowledge, all the hand recognition systems available nowadays on the market are 2D. However, during the last few years, there was a big research that has brought a few new methods and shown some possibilities of dealing with 3D hand geometry information that can be used for recognition of people.

A few most promising approaches are presented in the following subsections.

Finger 3D profile based approach

This method was proposed by Vivek Kanhangad, Ajay Kumar and David Zhang in [9]. Their system uses contactless 3D scanner to obtain the 3D information of the hand surface. It does not define any restrictions regarding hand positioning either. For the recognition itself, they use a fusion of 2D and 3D hand biometric features, which turn out to significantly improve the resulting performance of the system.

First, let us shortly describe the acquisition device they use. It is the commercial 3D laser scanner Minolta Vivid 910. Thank to this device, they are able to do the reconstruction in a contactless way, without requiring the user to place his hand somewhere and touch any surface. However, on the other hand, such a scanner is not really a cheap device and also, the acquisition system requires the background behind the hand to be of the black color. Because of these two restrictions, the final solution is not a candidate for future commercial use.

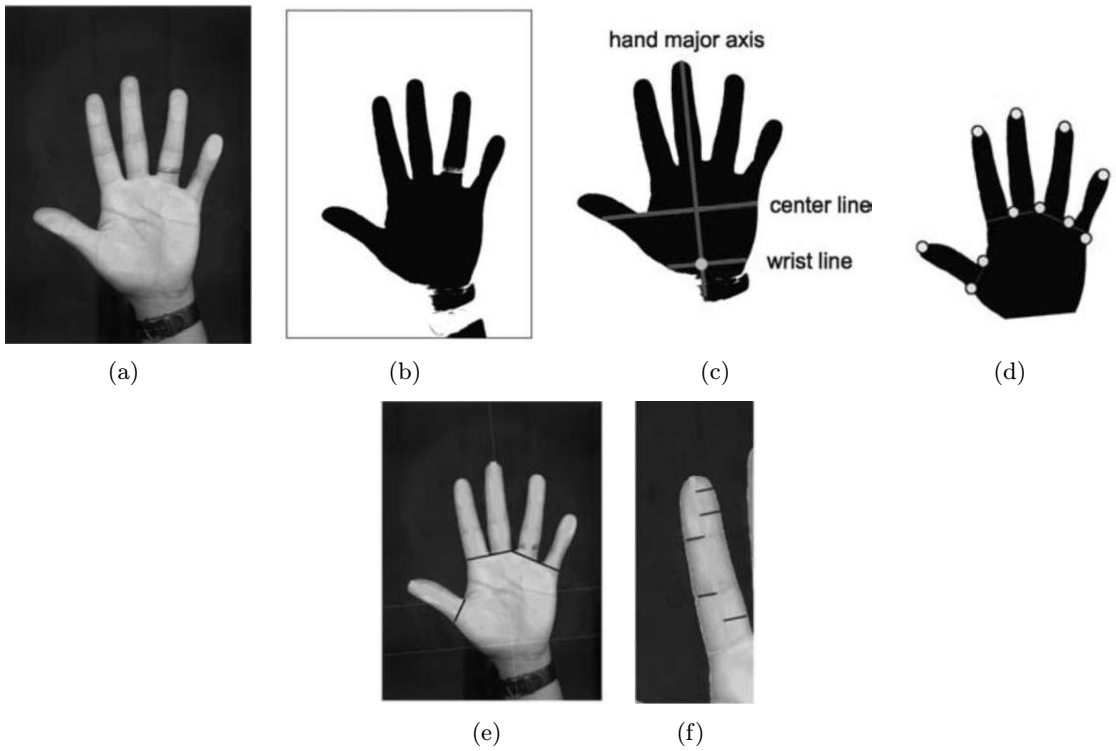


Figure 2.9: (a) Original image; (b) Result of segmentation; (c) Estimated axes of the hand; (d) Finger tips and finger valleys; (e) Separated hand fingers; (f) Distances from the finger center axis [2].

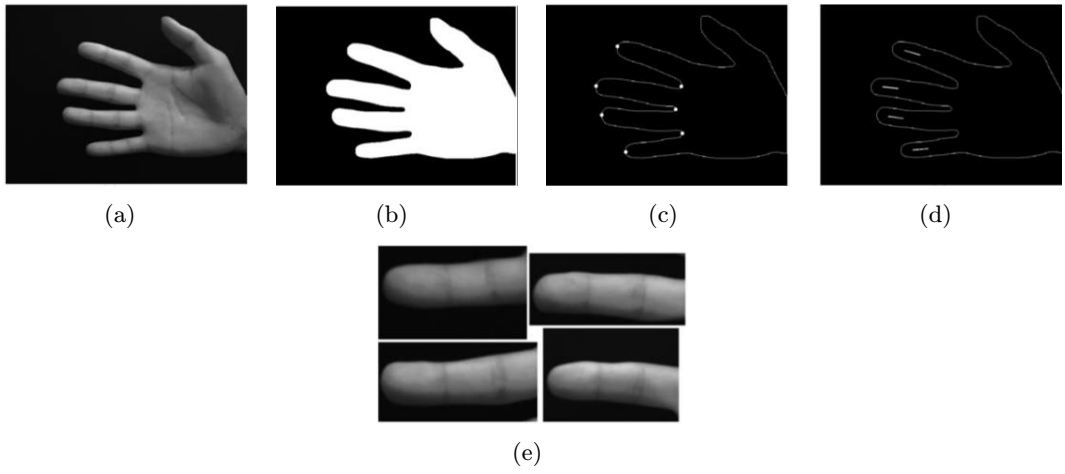


Figure 2.10: (a) Intensity image; (b) Result of segmentation; (c) Finger tips and finger valleys; (d) Finger orientations; (e) Extracted separate fingers [9].

The feature extraction is as follows. Input intensity images are first thresholded and then hand contour is extracted. Next, finger tips and finger valleys are located in the contour image. Using the detected points, orientation of each finger is estimated and separate fingers

are extracted from the image. The stages are depicted in Figure 2.10. Based on the fingers extracted from the intensity images, finger can be extracted also from the range images and visualized in 3D, as shown in Figure 2.11(a). For each finger, several of cross sectional segments are obtained at uniformly spaced distances along the finger length, according to in [9]. These segments are shown in Figure 2.11(b). Next, mean curvature and unit normal vector are computed for each data point on the particular segments, see example in Figures 2.11(c), 2.11(d) and 2.11(e). Based on these values, the final feature vector is created.

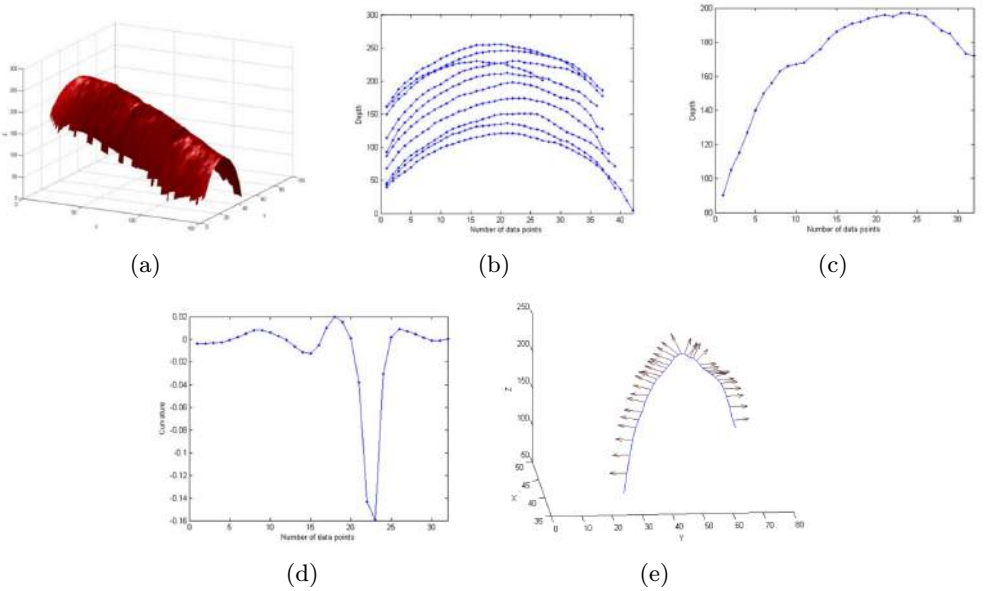


Figure 2.11: (a) Acquired 3D finger model; (b) Finger cross sectional segments; (c) 2D cross sectional segment; (d) Segment curvature; (e) Segment unit normal vectors [9].

The recognition is based on the comparison of the aforementioned feature vectors. The final score is created as a combination of results of matching the features on each finger separately. Curvature features matching for each pair of corresponding fingers is based on cosine similarity, as shown in [9]. Matching of unit normals is performed in a similar way. Details of the method are described in [9].

3D palm structure

An advantage of this method is the pose invariance and the fact that it can be used in the touchless scenario. Hence it is even more user friendly than classic 2D hand approach. Similarly to the previous approach, it uses 2D hand geometry and 3D hand geometry mentioned in the previous section, but on top of that, it also adds 3D and 2D information in the palmprint.

First, the hand has to be detected in the intensity and range images. Detection is based on the intensity image information and the approach is the same as the one presented in the previous section.

Hand pose correction is done the following way. First of all, palm center is detected using distance transform [10]. Distance transform appears to be more robust than approaches based on finger tips and valleys if we consider that it can be quite difficult or, in particular

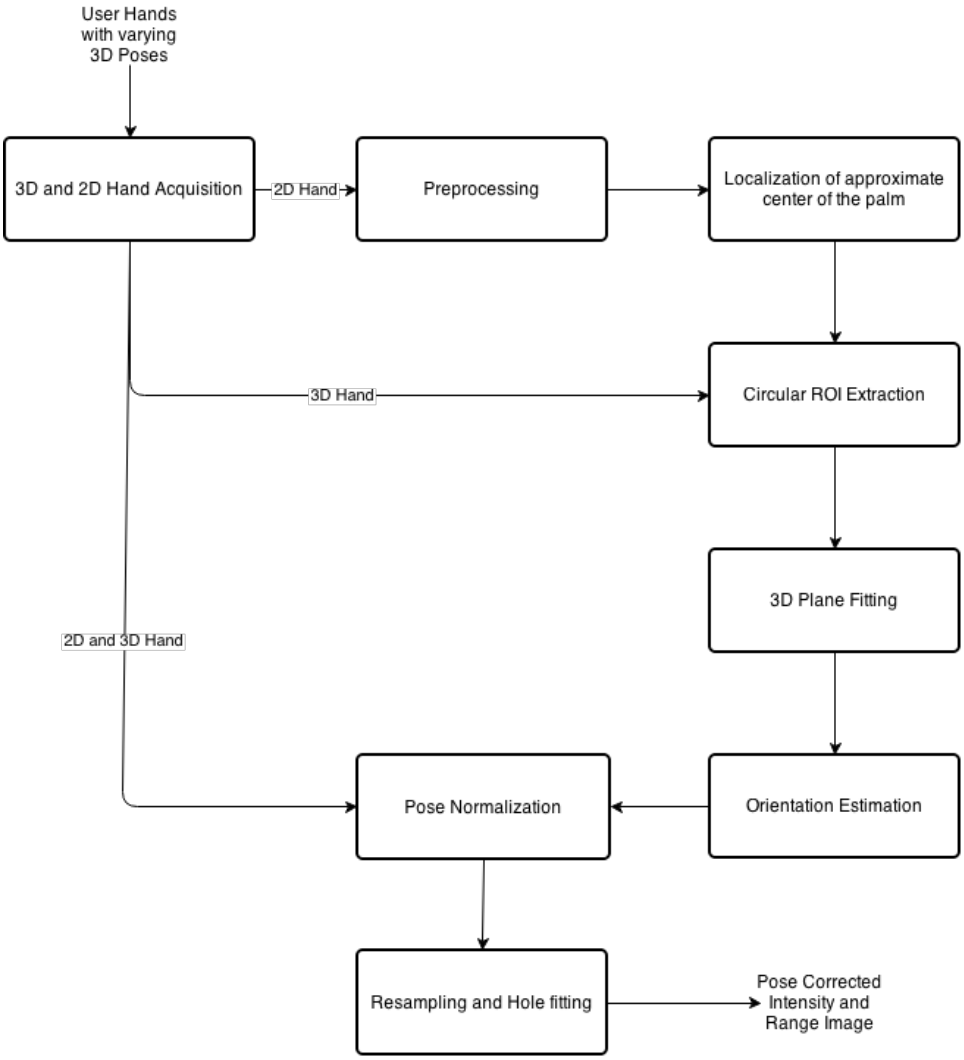


Figure 2.12: Pose correction processing pipeline [10].

cases even impossible, to detect finger valleys when the hand is rotated by a bigger angle. A circular region around the detected center is then taken. This region is approximated by a plane, which is fit using the iteratively reweighted least squares (IRLS) technique [10], which allows to assign a weight to each data point. The weight defines how far the point is from the fitted plane. Therefore the IRLS approach that they employed is less influenced by the outliers in the data, which may in their case arise from the bend or the deformations of the hand. Next, the normal to the estimated plane is computed. The normal describes the orientation of the hand in the 3D space. Using obtained information, pose correction is carried out and the results are pose corrected 3D points that have to be converted into range and intensity images for further processing. The whole process is described in detail in [10]. The whole correction process is illustrated in Figure 2.12.

Feature extraction is based on the the pose-corrected intensity and range images and includes [10]:

- **3D palmprint** - it is extracted from the range images of the hand from the palmprint

ROI (Region Of Interest). These features, mainly depth and curvature of the palm lines and wrinkles, are highly discriminatory;

- **2D palmprint** - identification using 2D palmprint has already developed in the past and there are many methods. In this case, approach based on Gabor filtering is used;
- **3D hand geometry** - same features as described in the previous section are used;
- **2D hand geometry** - this approach is already widely used in the industry. For the available methods, see Section 2.2.1.

The complete system is done as a dynamic fusion based on the weighted sum rule in order to combine match scores of each individual method into the final score. Block diagram of their system is shown in Figure 2.13.

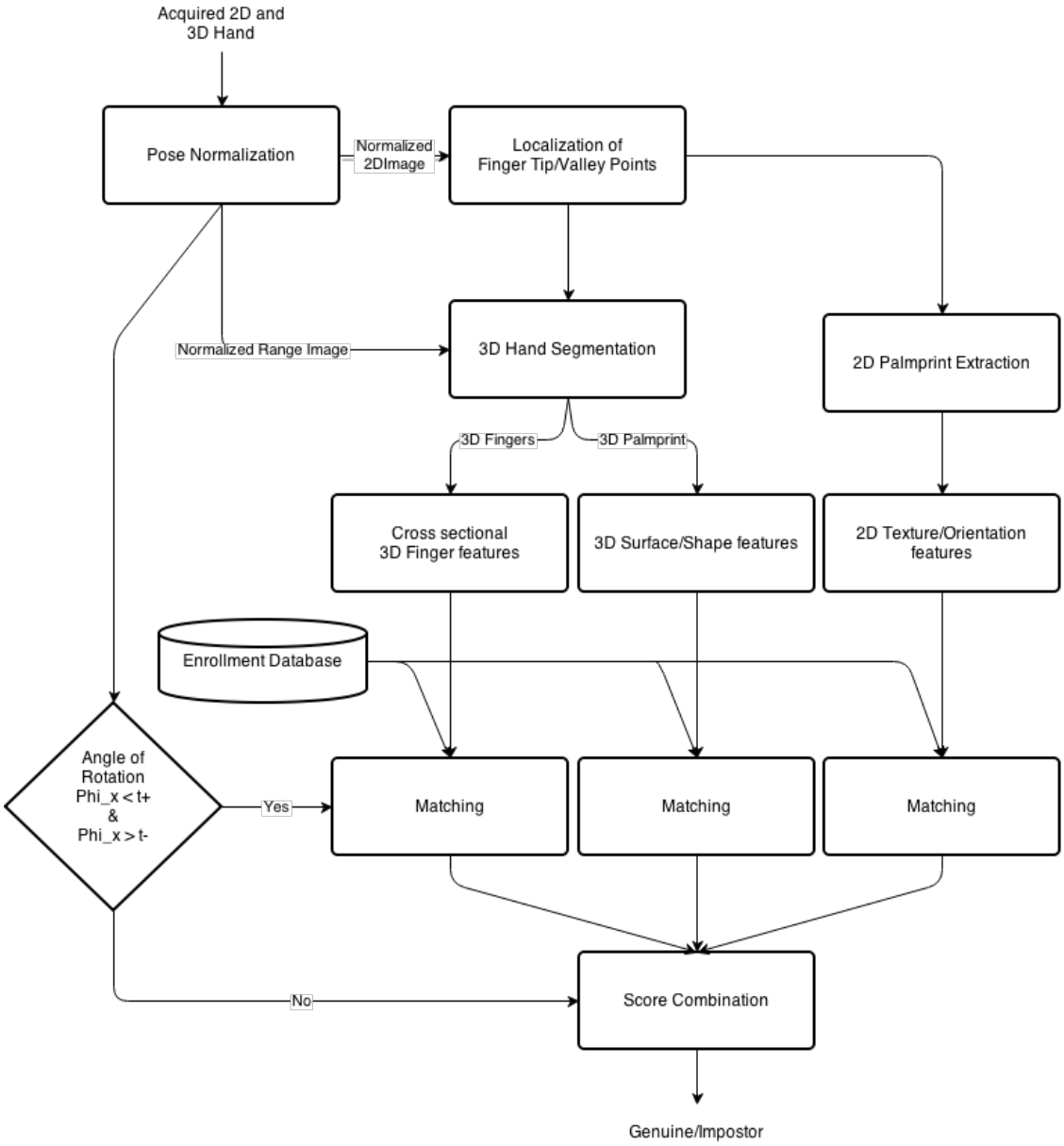


Figure 2.13: Block diagram of the proposed system based on the fusion of multiple biometric properties [10].

2.3 Multimodal biometric systems

As stated in [17], humans recognize one another using information presented by multiple biometric characteristics. It would be much harder for people to recognize others using only one biometric characteristic, but when the characteristics are put together, it makes it much easier and more reliable.

This is true also for the the biometric systems. Combining more biometric characteristics together results in a higher reliability. The main advantages of the multimodal biometric systems are in the following list:

- substantial improvement in the matching accuracy of the biometric system;
- flexibility of use due to the fact that more biometrics are enrolled in the system;
- it is more difficult for an impostor to spoof multiple biometric traits;
- reduction of the impact of noisy data from some biometric sensors under specific conditions;
- higher fault tolerance of the biometric sensors - if one sensor is broken, there are other that can be used.

More detailed description of the advantages can be found in [17] as well as much more detailed introduction into the problematics of the multimodal biometric systems in general including also the issues that have to be faced when designing such systems.

One of the main issues that has to be faced during the design of the multimodal biometric system is how to create the final match score by combining match scores from all the available sources. This is further described in the following subsection.

2.3.1 Matching score fusion

In order to be able to combine match scores obtained from different feature vectors, it is important to consider the meaning of the score for each particular system. That means, whether the higher score equals higher similarity or the opposite. Also, it has to be taken into account that the scores from the different biometric systems are usually in different ranges. Therefore a method to normalize the score values so that they can be easily combined together has to be used.

The complete list of the possibilities and their mathematical description can be found in [17]. One of the possible solutions is to use the score normalization methods. That is, changing the location and scale parameters of the match score distributions at the outputs of the individual matchers so that the match scores of different matchers are transformed into a common domain as stated in [17].

All the parameters used for the normalization can be estimated either using fixed training set (so-called fixed score normalization [17]) or they can be estimated based on the match score of the current sample (so called adaptive score normalization [17]).

As written in [17], for a good normalization scheme, the estimates of the location and scale parameters of the match score distribution must be robust and efficient. **Robustness** is the insensitivity to the presence of outliers. The **efficiency** is the proximity of the obtained estimates to the optimal estimates when the distribution of the data is known.

There are several normalization techniques. Probably the simplest normalization technique is called **Min-max normalization** and it is described by Equation 2.1

$$ns_j = \frac{s_j - s_{min}}{s_{max} - s_{min}} \tag{2.1}$$

where s_{min} refers to the minimal obtained match score, s_{max} is the maximal obtained match score, s_j is the match score of the j^{th} sample and ns_j is the normalized score of the j^{th} sample.

Another method is called **Z-score normalization**, which works with the mean and the standard deviation values obtained from the training data samples. It is described by Equation 2.2

$$ns_j = \frac{s_j - \mu}{\sigma} \tag{2.2}$$

where μ is the mean value, the σ is the standard deviation and ns_j and s_j have the same meaning as before.

Next one is for example so-called **Median and MAD normalization**, which uses the median of the matching scores and MAD (Median Absolute Deviation) of the matching scores, which is defined as

$$\text{MAD} = \text{median}\,|s_j - \text{med}|, \tag{2.3}$$

where the median is the median function, s_j is the matching score of the current sample and med is the median of all the training samples. Having MAD defined, the normalized match score can be easily computed using Equation 2.4

$$ns_j = \frac{s_j - \text{med}}{\text{MAD}}, \tag{2.4}$$

where ns_j is the normalized match score and the other values have the same meaning as in Equation 2.3 earlier.

Apart from the three methods mentioned above, there are many more like so-called **Decimal scaling**, **Double sigmoid normalization** or **Tanh estimators**. It is important to note that not all of the methods are both robust and efficient. The overview of the efficiency and robustness properties of the normalization methods is shown in Table 2.1. For more detailed description of all the methods, please refer to [17].

Table 2.1: Overview of the robustness and efficiency of the available match score normalization methods taken from [17].

Normalization Technique	Robustness	Efficiency
Min-max	No	High
Decimal scaling	No	High
Z-score	No	High
Median and MAD	Yes	Moderate
Double sigmoid	Yes	High
Tanh-estimators	Yes	High

2.3.2 Feacture vectors fusion

Another possible solution for a multimodal biometric system is to perform the fusion on the feature vector level. In this case, the fusion is done before computing the matching score and therefore there is no need to solve the matching score fusion problem. However, the feature vector combination is not an easy task. It has to be taken into account that both feature vectors most probably contain values in a different range as already mentioned before in this section. Thus, this issue has to be addressed either when the feature vectors are combined or when they are compared. Otherwise, computing the distance between the two feature vectors would be completely dominated by the largest component [12].

The source [12] further says that at least appropriate data scaling, for example data whitening and then applying the Euclidean distance measure, is one of the possible solutions. It is also stated that the combination of whitening and Euclidean distance is in the end exactly Mahalanobis distance measure. Data whitening is represented by the following equation:

$$w_{wh} = \Sigma^{-1/2} (w - \mu) \tag{2.5}$$

where w is a random vector with mean μ and covariance matrix Σ defined as $\Sigma = \sum_{(i,j)\in S} (x_i - x_j)(x_i - x_j)^T$, finally w_{wh} is the whitened version of the vector.

Distance metric learning

But [12] also mentions that such an unsupervised scaling is generally not good enough and in order to get a good performance, it is strongly suggested to use supervised learning of the feature weighting in order to transform the data from one space to another.

One of the possible solutions is to use the metric learning, where the goal would be to learn the transformation of the data based on the supervised information regarding the distances of the transformed data, as stated in [12].

The source [12] says that one of the both simplest and most popular approaches to learning metrics is so-called "Mahalanobis distance learning" which tries to learn distances of the form

$$d_A (x, y) = (x - y)^T A (x - y), \tag{2.6}$$

where A is a positive definite matrix. The matrix A can be easily viewed as applying a linear transformation of the input data. Because A is positive definite, it can be factorized as $A = G^T G$ and it can be easily derived that $d_A (x, y) = ||Gx - Gy||_2^2$.

One of the most popular methods for metric learning is the **Large-Margin Nearest Neighbors (LMNN)**. It combines relative distance constraints and regularizer of Xing. et al. [12]. The method finds A by solving the optimization problem

$$\min_{A \succeq 0} \sum_{(i,j)\in S} d_A (x_i, x_j) + \lambda \sum_{(i,j,k)\in R} [1 + d_A (x_i, x_j) - d_A (x_i, x_k)]_+ , \tag{2.7}$$

where $\lambda \geq 0$ is a parameter, the notation $[z]_+ = max(0, z)$ is used for the standard hinge loss and S is the set containing all the pairs (i, j) of target neighbors and the set R on the other hand is the set of all the triplets (i, j, k) such that x_i and x_j are the target neighbors and x_k is a point with a different label.

The goal of the method is that a given data point should have the same label as its nearest neighbors and the points with different labels should be as far from that point as possible. The aim of the LMNN approach shown in Figure 2.14 is therefore to minimize the number of impostors using the relative distance constraints as described in [12].

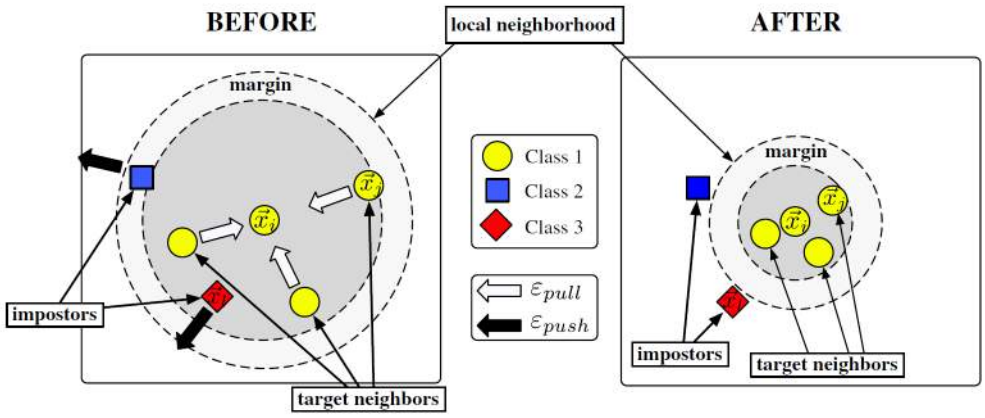


Figure 2.14: LMNN metric learning principle visualization.[12]

2.4 Evaluation of biometric systems

The aim of this section is to introduce the basic terminology and methods that are used for the evaluation of the biometric system performance. The first subsection explains the used terminology to the reader. After that, the most commonly used performance measures are introduced. In the end of this section, several options for visualization of the performance are shown.

2.4.1 Basic terminology

As stated in [2], the following three basic types of classification are considered speaking about biometric systems.

- **verification** - the task is to determine, whether the presented set of biometric traits belongs to a previously marked user;
- **identification** - apart from verification, here the task is to determine to which user in the database the presented set of biometric traits belongs;
- **recognition** - the goal is to say to which class the semantic content of the biometric traits belongs.

In the ideal case, the error of the just presented types of classification is 0. But that is never the case in the real world applications. Therefore there is a need to have particular measures that describe how good the biometric system is.

Speaking about biometric traits in particular, there are a few very important terms that have to be considered. Those terms are shortly described next.

The first is **intraclass variability**, which describes the changes in the traits that occurs during multiple acquisitions of the same user. These changes can arise due to the current physical state of the individual, his mental state or just due to the changes in the surrounding environment. This variability is never desired and in ideal case, it would be 0.

The second term is called **interclass variability**. Apart from intraclass variability, the interclass variability stands for the differences in a particular biometric trait for different users. This measure is always desired to be as high as possible. If it is not high enough,

the particular trait is not a good candidate to be used for the recognition between different individuals. This term is closely related to the **biometric entropy**, which says how much information does the particular biometric trait contains. The higher the biometric entropy is, the higher the interclass variability is.

In case of verification, the biometric system can give wrong results, which can be basically divided into the following two groups, as listed in [2].

- **false rejection** - rejection of the right hypothesis (the user had the right to enter the system, however he was rejected);
- **false accept** - acceptance of the wrong hypothesis (the user had no right to enter the system, however he was accepted).

The just introduced terms can be expressed using multiple measures that are explained in the next section.

2.4.2 Comparison error measures

One of the most important things regarding the behavior of a biometric system is the so-called decision threshold. The **decision threshold** is a value that divides the matching scores into two groups. The threshold defines which matching scores are interpreted as matches and which ones as non-matches. The meaning of the decision score is illustrated in Figure 2.15.

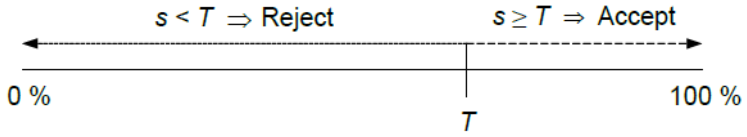


Figure 2.15: The result is either accept or reject depending on the current matching score value s and how the decision threshold T is set [2].

When the matching score is compared with the decision threshold, the biometric system returns a result that can be either right or wrong decision. The result of the biometric system for the verification (identification) can be one of the following:

- **True Accept** - the person A is accepted as A;
- **True Reject** - the person A is rejected as B;
- **False Accept** - the person A is accepted as B;
- **False Reject** - the person A is rejected as A.

For estimating the error measures in the biometric systems, mainly the False Accept and False Reject cases are important. Based on those values, the following measures can be computed.

The first measure is **False Accept Rate** (FAR [2]), and it represents the amount of the verification attempts with false biometric traits presented that are accepted by the biometric system. It is the probability that the biometric system will classify two different

biometric samples as the same and thus fail to reject a possible intruder. FAR can be computed as

$$FAR = \frac{\text{dmc}_{true}}{\text{dmc}_{all}} \tag{2.8}$$

where dmc_{true} stands for the number of comparisons of different samples with result match and dmc_{all} is the number of all the comparisons of different samples.

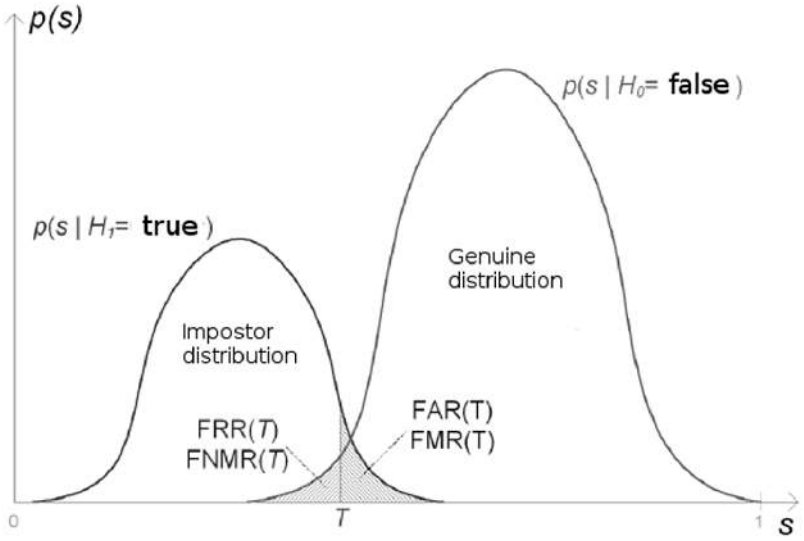


Figure 2.16: An example of the genuine and impostor score distributions with the FAR and FRR marked [2].

Another very important measure is the so-called **False Reject Rate** (FRR [2]) that represents the amount of the verification attempts with true biometric traits presented that are rejected by the biometric system. Therefore it is the probability that the biometric system will classify two biometric samples from the same person as different and thus fail to accept a user that is enrolled in the system. FRR computation is represented by the following equation

$$FAR = \frac{\text{umc}^A_{false}}{\text{umc}^A_{all}} \tag{2.9}$$

where, the umc^A_{false} is the number of comparisons of samples from user A with result non-match and umc^A_{all} is the number of all comparisons of samples from the user A.

The last measure mentioned in this text is **Equal Error Rate** (EER [2]), which is theoretically defined by condition FAR = FRR and in the real applications it is the part of the distribution plots, where the both FAR and FRR equals. If the decision threshold T is set to the EER value, the number of false accepts and false rejects will be the same. As said in [2], it is therefore possible to set the decision threshold T according to the needs of the particular application.

Apart from FAR, FRR and EER there are many more measures (e.g. FMR, FNMR, FTE, FTA, etc.) that are described in more detail in [2].

2.4.3 Performance visualization

The performance of the system is usually visualized using the so-called **Receiver Operating Curve** (ROC). This curve is derived from the dependency of the FAR and FRR values on the decision threshold value. With the change of the decision threshold, one of the FAR and FRR values is increased and the other one decreased. Both values are changed together, but always in the other direction - when one increases, the other decreases). The ROC curves represent the detection ability of the FAR in relation to FRR, as stated in [2]. An example of the ROC curve is shown in Figure 2.17.

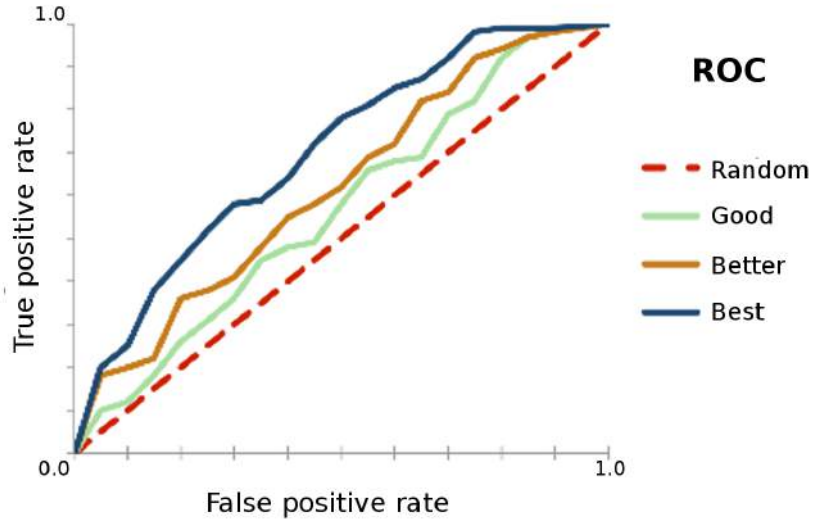


Figure 2.17: An example of the ROC curve [2].

As you can see from the Figure 2.17, the better the performance of the system is, the closer the ROC curve gets to the top left corner of the graph.

Another alternative to the ROC curve might be the so-called **Detection Error Tradeoff** (DET) curve, which is basically the same, just with a different values on the axes. For more information about the DET curve, please refer to [2].

2.5 Industrial systems

So far, there are only 2D geometry based biometric systems available on the market. No 3D geometry based system has been released yet.

As listed in [2], the first 2D hand recognition device was deployed in 1970's under the brand **Identimat**. From that time, it was used in many industrial applications. In a few cases, it is still in use, only with some modifications according to the development of the 2D hand geometry systems.

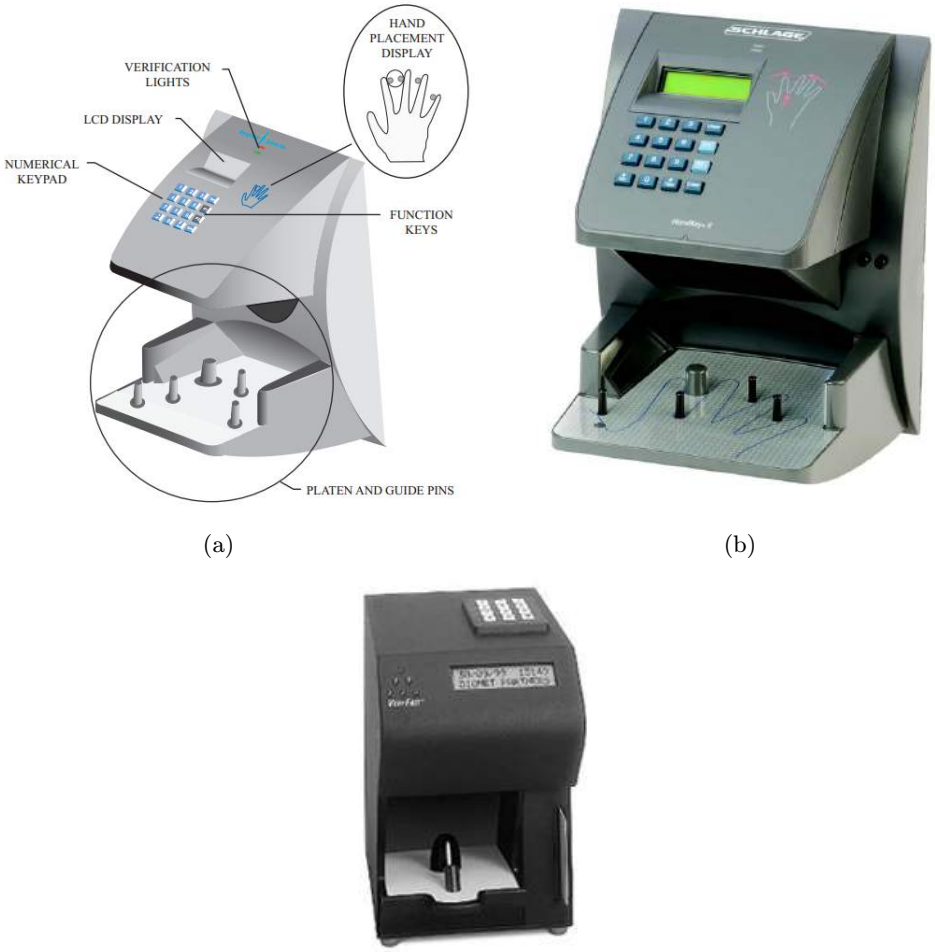


Figure 2.18: HandKey II biometric system: (a) schematics of the HandKey II; (b) HandKey II real photo; (c) Digi-2 biometric systems by Biomet Partners company [2].

Another commercially offered device is **HandKey II**, shown in Figure 2.18(b), which is currently used as a part of the security system in the Dukovany nuclear power plant in the Czech republic. The device uses low-level infrared light and CMOS camera to capture images of the hand. Mirrors are mounted in the device on the sides of the platen where the hand has to be placed. Thanks to those mirrors, a pseudo-3D information can be obtained by observing the hand not only from the top, but also from the sides (the reflections in the

mirrors). Input images are converted into nine byte template and stored in the database. Device can operate either as a standalone unit, in a network with other HandKey devices or in a network with a computer.

Another system that is worth mentioning is called the **Digi-2** (Figure 2.18(c)) and was released in 1995. This device uses CCD camera to capture hand finger images. Fingers are captured from two directions so it is again possible to get a pseudo-3D information about the fingers. Features are extracted from the input images using a microprocessor, which is part of the device. The comparison can be done afterwards. Template size is 20 bytes.

Chapter 3

Proposed 3D hand geometry recognition system

As shown in the previous chapter in Section 2.2, a lot of research has already been done in this field. However, those systems usually do not pay so much attention to the final price and their usability in the real world applications. Because of that, all the research that has been done uses either very expensive acquisition device or it arises some constraints that makes it impossible to be used commercially. Apart from that, the system presented in this section uses acquisition devices that can be obtained at reasonable prices and it does not tend to define any constraints that would be problematic to reach in the real world applications. To compensate the problems that arise from not having many constraints, it uses fusion of both 2D and 3D hand geometric information for the recognition task.

3.1 Acquisition device

As 3D sensor, the soon to be released Intel RealSense camera shown in Figure 2.6 was used.

The RealSense camera is based on the time-multiplexed structured light projected in the near-IR spectrum.

The depth information comes in 640×480 resolution at up to 60fps. The camera has a tiny form factor (3.5mm thickness) and is intended to be embedded into laptop computers.

3.2 Input data acquisition

The input data are obtained using the 3D camera described in the previous section. This section describes which images are collected during the capture process and how they are stored.

Capture process output

There are five output data files that are created with each successful capture. Namely, those are:

- **intensity.png** - intensity image captured by the IR camera;
- **depth.png** - depth map obtained by the 3D camera;

- **mask.png** - mask of the image region that contains hand created from the depth map;
- **vertices.yml** - hand surface 3D model vertices stored in YAML file format for easier serialization/deserialization;
- **model.ply** - hand surface 3D model vertices stored in PLY format.

Example of the three output images plus the visualized point cloud are shown in Figure 3.1.

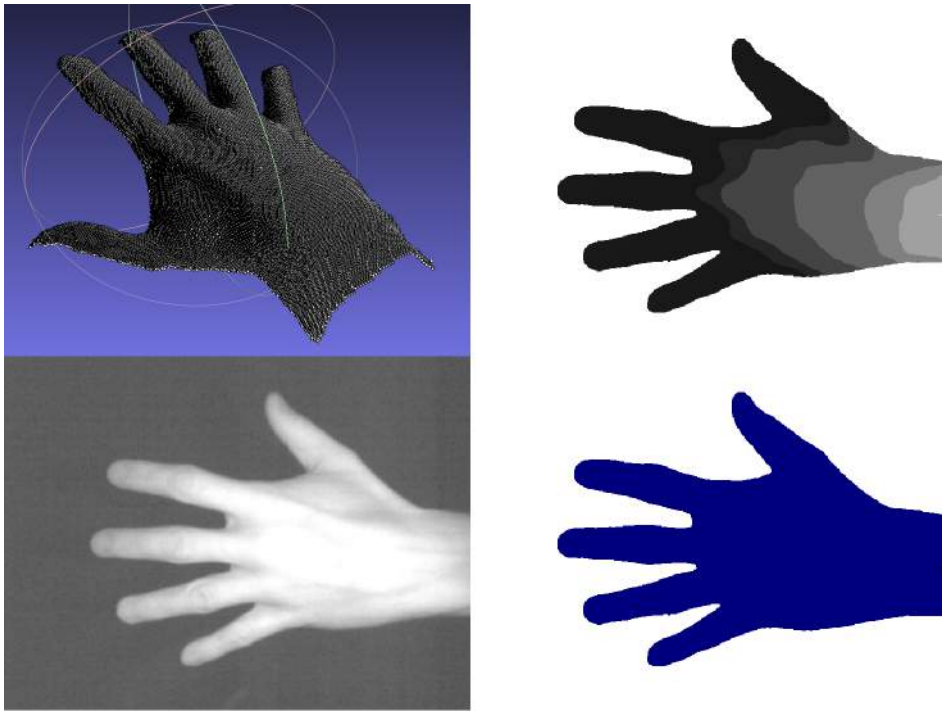


Figure 3.1: System input data. Top left: visualized point cloud; Top right: depth map; Bottom left: IR camera intensity image; Bottom right: hand mask.

Hand positioning

The most important part during the input data acquisition phase is to define the constraints that has to be satisfied in order to get a sufficient input data. This can be done by either only verbally instructing the users how to place their hand before they start using the device or by combining those verbal instructions with a positioning application, in which the program tries to navigate the users to reach the correct positioning of the hand in the scene.

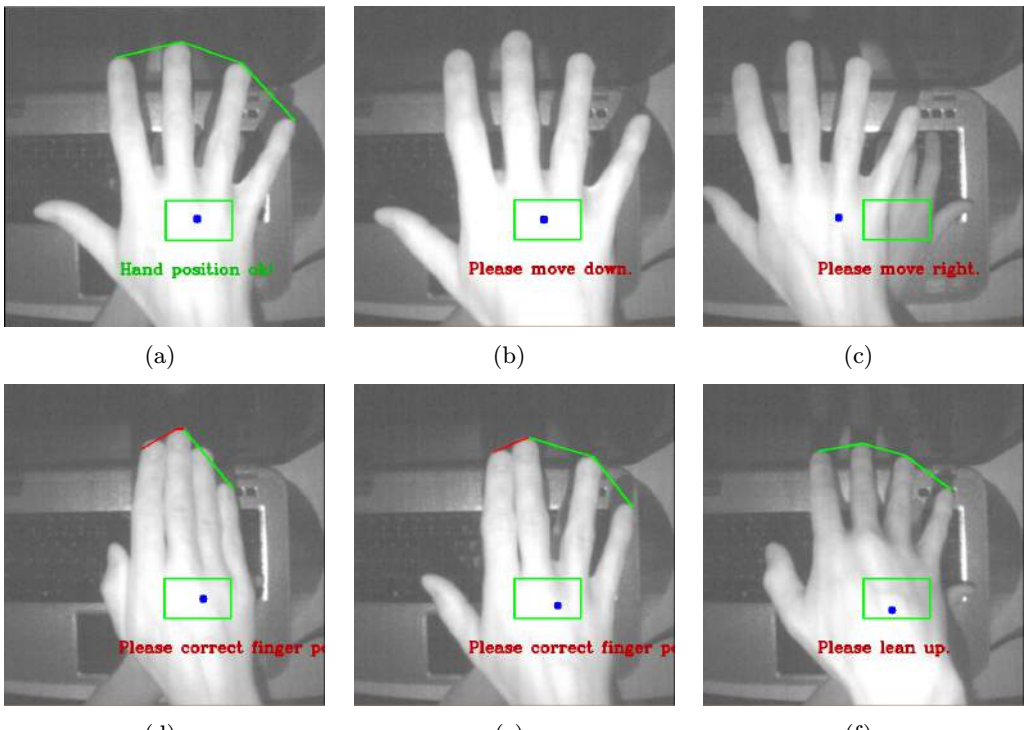
The proposed system chooses the latter approach. Therefore, the constraints can be divided into two groups. The verbal ones, which are not checked by the positioning application and the application constraints, which are checked automatically by the input data acquisition application.

The **verbal constraints** arise from how the device is designed to be used. Those constraints are:

- **hand dorsal side constraint** - defines that the hand dorsal side, instead of the hand palm, has to be presented to the camera;
- **fingers constraint** - defines that the fingers have to be presented stretched with some tolerance, which means that they can be bend, but just to the extent that the user feels comfortable when presenting the hand to the device;
- **hand axis weak constraint** - defines that the axis that goes through the middle finger fingertip and is perpendicular to the wrist line has to be approximately perpendicular to the optical axis of the 3D camera.

In order to use the device correctly, satisfying all three listed constraints is a must.

When all verbal constraints are met, the positioning application is expected to work correctly and navigate user to place his hand the proper way in the scene. To achieve the proper hand placement, another set of constraints has been created. Those are so-called **application constraints** and they define how to place the hand into the scene so that the positioning application accepts the position and does the input data capture. List of the application constraints follows.



(a) (b) (c)

(d) (e) (f)

Figure 3.2: Hand positioning application constraints: (a) Well positioned hand; (b) Mean depth constraint violated; (c) Centroid position constraint violated; (d) Fingertip count constraint violated; (e) Fingertip distance constraint violated; (f) Hand axis strong constraint violated.

- **mean depth constraint** - defines the accepted distance of the hand surface from the 3D camera by checking the mean depth of the hand surface that can be obtained from the depth map image (Figure 3.2(b));

- **centroid position constraint** - defines the required rectangular area in which the hand centroid must lie (Figure 3.2(c));
- **fingertip count constraint** - defines that exactly five fingertips have to be detected on the hand surface in order to be accepted by the application (Figure 3.2(d));
- **fingertip distance constraint** - defines that the distances between little to ring, ring to middle and middle to index fingertips have to be approximately the same (Figure 3.2(e));
- **hand axis strong constraint** - it is the same as **hand axis weak constraint**, but in this case the orientation is checked against allowed range by the application (Figure 3.2(f)).

To satisfy all the application constraints, the user is led towards the good hand position by a few markers displayed on the screen during the positioning. As shown in Figure 3.2(a), the markers are the green rectangle that defines where the hand centroid should lie, the blue dot that marks the computed hand centroid, the lines between the fingers that indicate whether the fingers are spread enough (green lines) or some of them are too close to each other (red lines) and also the text indication that shows user the current status of the hand positioning.

When the hand position is correct, capture of the data could be done either automatically or manually by clicking a button on the keyboard. The decision whether to do the capture automatically or not depends on a particular use case.

3.3 Hand biometric features

Human hand biometric features used in this system can be basically divided into two groups according to the dimensionality of the space they are observed in. These groups are namely the 2D features, which are observed in the 2D space and the 3D features, which can be measured in the 3D space. For both of these groups, it is a proven fact that they provide sufficient amount of information for the biometric recognition of people. Both groups are described in more detail in the following subsections.

2D features

The biometric features measured in the 2D space are usually based on the analysis of the hand contour extracted from a hand image. The list of basic measures that are usually obtained from 2D image of the human hand in the industrial applications was presented earlier in Section 2.2.

In this particular solution, the following distances were measured over the hand surface:

- **finger length** - length of the finger is measured as a distance between fingertip and center of the finger valleys (Figure 3.3(a));
- **finger valleys distance** - distance between the finger valleys of a particular finger (Figure 3.3(b));
- **finger widths** - widths of a particular finger in predefined positions along the finger axis (Figure 3.3(c)). The predefined positions are approximate locations of the finger joints;

- **wrist width** - width of the hand wrist (Figure 3.3(e));
- **wrist - valley distances** - distances between the wrist points and the finger valley points (Figure 3.3(d)).

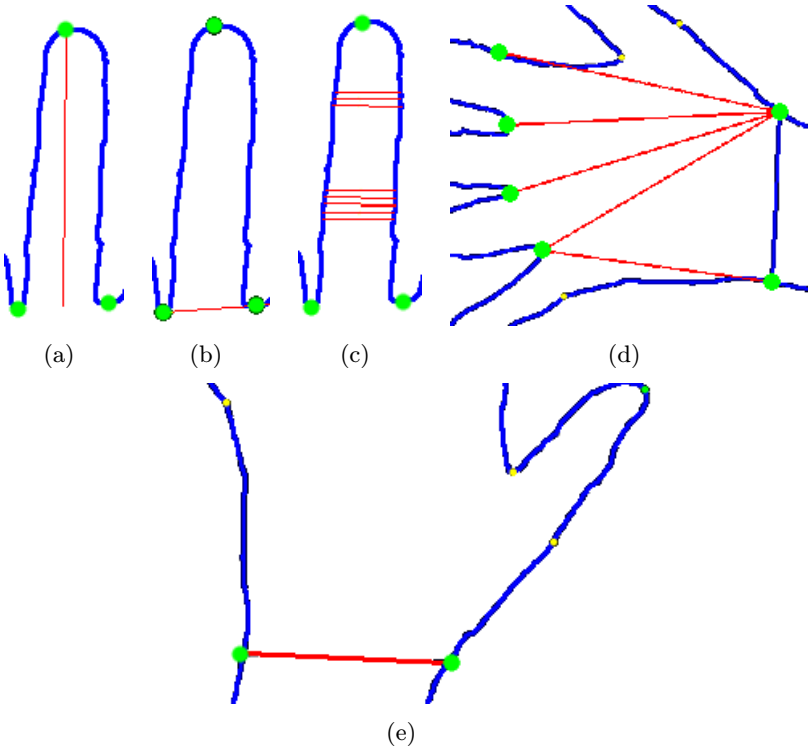


Figure 3.3: Measured 2D features: (a) Finger length (red line); (b) Finger valleys distance (yellow line); (c) Finger widths (yellow lines); (d) Wrist - valley distances (red lines); (e) Wrist width (purple line).

3D features

Considering the 3D space, hand model contains also the depth information. In the 3D space, all the distances measured in the 2D space could be measured again. However, in this case, it is not just computing a distance between two points, but the distance is computed by traversing through each point on a specific line over the surface. This way, more robust measure that does not suffer so much from the hand transformations can be obtained. Apart from the distances, it is also possible to get new features like normal vectors or curvatures.

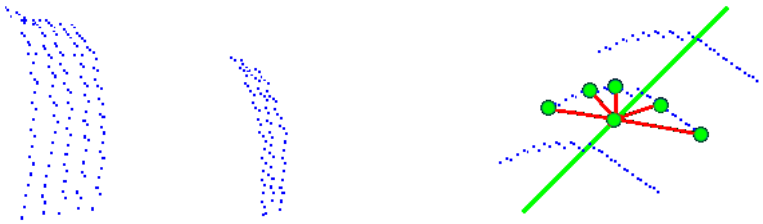
The 3D features can be obtained by analyzing either the depth map image or the vertices of the hand point cloud. This solution uses the latter, so the 3D features are obtained by performing analysis of the hand point cloud.

In this work, the following set of 3D features is used:

- **3D finger widths** - widths of a particular finger (a cross sectional segments) in predefined positions along the finger axis. The predefined positions are as well as in

case of 2D finger widths the approximate locations of the finger joints. The difference is that in the 3D case, width is measured by traversing all the points along the line that is perpendicular to the finger axis (Figure 3.4(a));

- **finger segment axis - surface distance** - distance between finger axis at the cross sectional segment and the points on the finger surface. The finger axis at the cross sectional segment is computed using two neighboring cross sectional segments (Figure 3.4(b)).



(a) (b)

Figure 3.4: Measured 3D features: (a) 3D finger widths; (b) 3D finger segment axis - surface distance (red line is the axis, green lines are the distances measured between the blue points).

3.4 Feature extraction

As the system is based on a 3D camera, some phases of the hand detection can be significantly simplified by taking advantage of the depth information. The most important steps of the feature extraction are described in the next few subsections.

Hand detection

This is always the first phase of the feature extraction algorithm. To be able to extract good features, hand object has to be detected in the scene image as precisely as possible. Apart from the previously used approaches, where the hand has to be placed in front of a dark or very reflective homogeneous surface in order to be detected precisely, this new approach does not have any requirements on the capturing environment at all.

As mentioned previously in Section 3.2, one of the input images in this solution is the depth map that can be used easily to separate the background from the foreground using the depth information (Figure 3.5(a)). A range of accepted depth values is specified and only the depth values that satisfy the following equation $z_{min} \leq z \leq z_{max}$, where z is the depth value of a particular pixel, and $[z_{min}, z_{max}]$ is the range of the accepted depth values, are marked as the hand surface (Figure 3.5(b)).

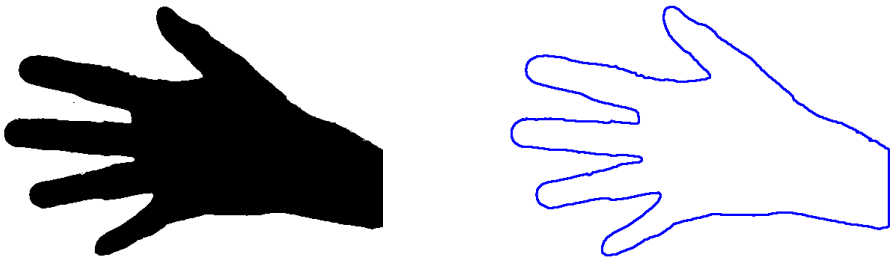
The proposed foreground - background separation method can produce some outliers. Those are removed in the next step.

(a) (b)

Figure 3.5: Foreground - background separation using the depth information: (a) Input depth map; (b) Output hand mask.

Contour detection

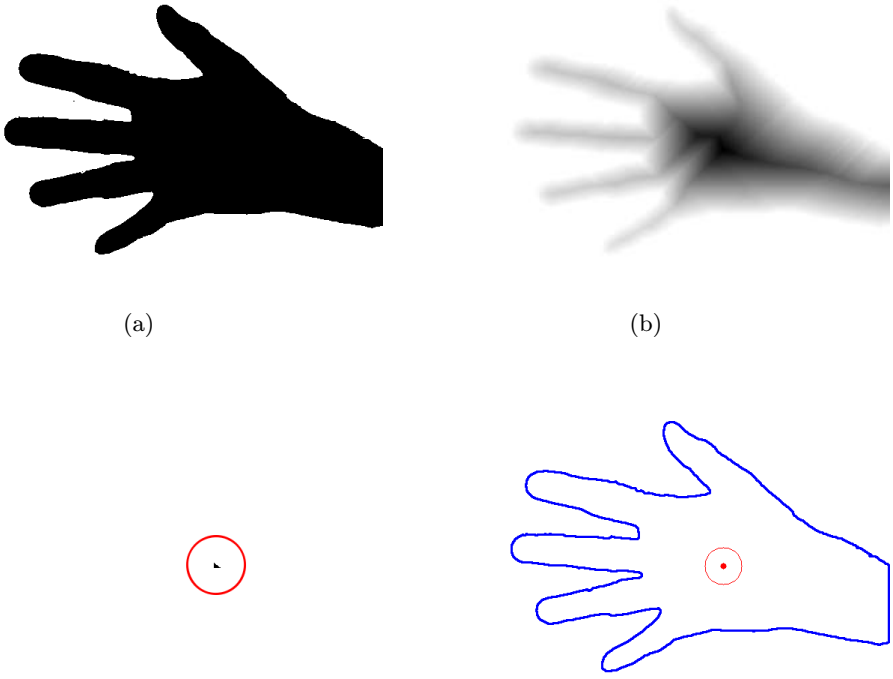
When the hand region is detected in the input images, the next step is to obtain the contour of the hand that will serve as an input for all the next processing steps.



(a) (b)

Figure 3.6: Contour detection using hand mask: (a) Input hand region mask; (b) Output hand contour.

As an input, this step takes the hand mask that was created in the previous step (Figure 3.6(a)). At the beginning, all the contours in the mask image are detected. To remove the possible outliers, the area is computed for every detected contour. The following equation $\text{◭} \geq \text{◭}_{min}$, where ◭ is size of the contour that is currently checked and ◭_{min} is the smallest size of the contour that is accepted, is used to decide, whether the contour is a hand contour or not. If the equation presented above is not satisfied, the contour is considered as an outlier, otherwise the contour is marked as a hand contour. The resulting detected contour might look like the one in Figure 3.6(b).



(a) (b)

(c) (d)

Figure 3.7: Hand centroid detection: (a) Input hand region mask; (b) Hand region after distance transform; (c) Thresholded result of the distance transform; Visualization of the detected hand centroid.

Hand centroid detection

Detection of the hand centroid is fundamental, because it affects the detection of the other important points such as the fingertips, the finger valleys and the wrist points. That implies a need for robustness of the chosen centroid detection method. The approach used in this solution is inspired by article [10]. Based on the article, distance transform is used in order to detect the center of the hand.

As an input, the hand mask image is used. This image is processed by the distance transform to get the distance of each point on the hand surface to the edge of the surface (Figure 3.7(b)). The pixels with the highest values are the ones lying in the center of the hand. Central hand area (Figure 3.7(c)) is then defined as all the points that satisfy the following equation $i \geq T$, where i is the distance transformed intensity value of a particular point in the result of the distance transform and T is the decision threshold that distinguishes between the central and the non-central points.

To get one centroid point (Figure 3.7(d)), average x- and y-coordinates of all the central points are obtained and point lying on this coordinate is taken as the hand centroid. Since there are only few points in the central area obtained from the distance transform, taking point lying on the average coordinates is sufficient solution.

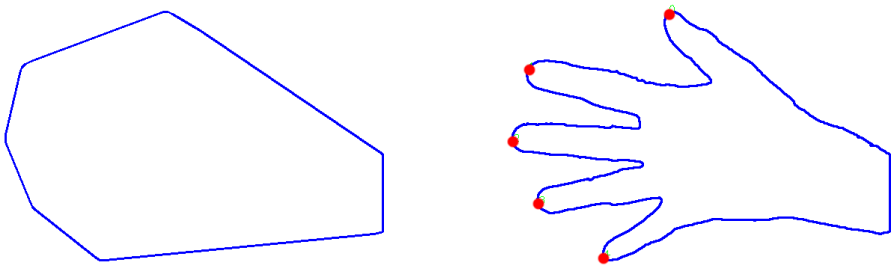
Fingertip detection

After the hand centroid is detected, the focus moves on to the fingertip detection. In this phase, prepared hand contour is finally used instead of hand mask image.

First of all, the hand contour is approximated by a convex hull (Figure 3.8(a)). Points of the hull are further filtered so that only those satisfying the following equation $x_0 \geq x$, where x_0 is the hand centroid x-coordinate, x is the x-coordinate of a candidate point that is being checked, are left as the candidates. During the filtering, basic check against surface curvature is done to ensure that the candidate point will not pass in case it lies on a straight part of the contour.

The group of the filtered candidates still contains a lot of points. However those points tend to group into clusters around the candidate fingertip positions. Because of that, points are divided into a few clusters by using an algorithm for splitting a set of N elements into equivalency classes as described in [16]. In each cluster, a point with the lowest x-coordinate in the cluster is taken as the candidate point and then its neighborhood is searched for the point with the highest curvature. The resulting point is marked as a fingertip (Figure 3.8(b)).

If there are more than five clusters, the five ones with the lowest x-coordinates are taken and the rest is discarded.



(a) (b)

Figure 3.8: Fingertip detection: (a) Hand contour approximated by convex hull; (b) Hand contour with detected fingertips.

Finger valleys detection

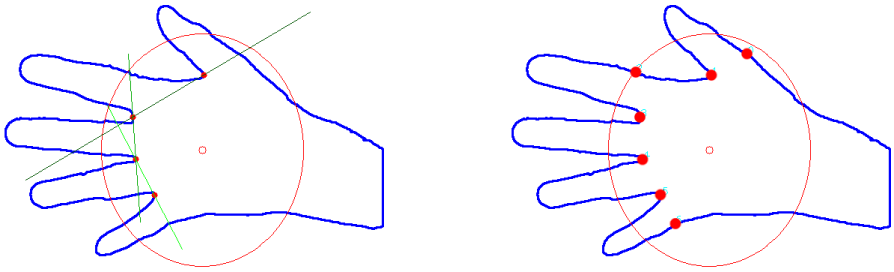
Detection of the finger valleys comes as the next step after the detection of fingertips. It is done in this order as the fingertips are required to detect the finger valleys.

All the five detected fingertips are taken and traversed from the first one to the last one. The part of the hand contour between each two neighbor fingertips is searched for the place with the highest x-coordinate. The detected points are marked as the finger valley candidates and further processed. In order to clarify that the point is really a finger valley, an elliptical area around the hand centroid is defined. The finger valley points must lie in the elliptical area. Therefore, one more check is performed using the following equation

$$\frac{x - x_0}{a^2} + \frac{y - y_0}{b^2} \leq 1 \tag{3.1}$$

where $[x_0, y_0]$ is the hand centroid and $[x, y]$ is the point to be checked. If the equation is satisfied, the point is marked as a finger valley, otherwise the point is discarded.

This way, the four main finger valley points are obtained. However, to be able to process all the fingers of the hand individually at a later stage, it is required to obtain seven finger valley points in total. Using this approach, each finger has two precise finger valleys. The missing points are namely left finger valley of the little finger, right finger valley of the index finger and right finger valley of the thumb.

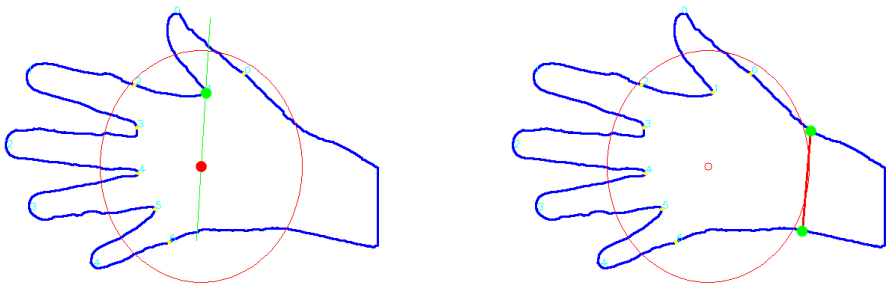


(a) (b)

Figure 3.9: Finger valleys detection: (a) Green lines used to define three more finger valleys after base finger valley detection; (b) Result of finger valleys detection.

To detect these points, an approach inspired by [2] is applied. Using the finger valleys that have already been detected, three lines going through two particular finger valleys can be defined as shown in Figure 3.9(a). By finding intersections of those lines and the hand contour, the missing finger valley points can be found.

In total, there are seven finger valleys that define the area of the contour belonging to a particular fingers. Results of the finger valley detection process can be seen in Figure 3.9(b).



(a) (b)

Figure 3.10: Wrist line detection: (a) Green line is the one used to obtain direction vector of the wrist line; (b) Result of wrist line detection.

Wrist line detection

The last thing that has to be detected to get the full description of the human hand is the wrist line. The wrist line is a line marking where the wrist of the hand is. To detect the wrist line, the previously obtained information are used.

Knowing the positions of the fingertips and having the hand centroid detected, an approximate axis of the hand going through the centroid and the middle finger fingertip can be drawn assuming that the direction vector from Equation 3.2

$$d^{axis} = t^{middle} - (x_0, y_0) \tag{3.2}$$

where d^{axis} is the direction vector and t^{middle} is the middle finger fingertip, is also known. It can be assumed that the wrist line goes through a point that lies in a particular distance from the hand centroid in the opposite direction than the middle fingertip, expressed by the following equation

$$p^{wrist} = (x_0, y_0) - s \cdot d^{axis} \tag{3.3}$$

where s defines how far from the hand centroid the wrist point is. Taking this information into account, it is now only needed to obtain the direction vector of the wrist line. From many observations of the human hands, knowing the positions of the finger valleys, it can be seen that the orientation of the wrist line corresponds to the orientation of the line that is going through the 2$^{\text{nd}}$ finger valley and the hand centroid point as visualized in Figure 3.10(a). Thus the direction vector can be obtained using the following equation

$$d^{wrist} = (x_0, y_0) - v_2 \tag{3.4}$$

where v_2 is the finger valley point with index two and d^{wrist} is the direction vector of the wrist line.

The wrist line is described by two points on the hand contour in the end. Those points are computed by intersecting a line defined by the previously mentioned point on the finger axis and the just obtained direction vector with the hand contour. The line can be expressed by the following equation

$$d_y^{wrist} x - d_x^{wrist} y - (d_y^{wrist} p_x^{wrist} - d_x^{wrist} p_y^{wrist}) = 0 \tag{3.5}$$

where p^{wrist} is a point that lies on the wrist line.

An example of the detected wrist line is shown in Figure 3.10(b).

3.5 Feature matching

The extracted features that were described in the previous section are compared using the methods presented in this section. First of all, the feature vectors that are created from the extracted features are described. Next, the comparison methods used for the 2D geometric features are described. After that, the methods used for matching of the 3D geometric features are introduced. At the end of this section, the fusion of both the 2D geometry matching score and the 3D geometry matching score is explained.

Feature vectors

To create a common representation of the extracted features, all the computed features are put into one dimensional feature vector that has a length of the number of measured values.

There are two feature vectors created for each input sample. A vector of the 2D geometric features and a vector of the 3D geometric features. Both vectors are basically composed of a specific amount of floating point numbers.

The **2D feature vector** is described first. It has the length of 41 values. All the values are expressed in pixel units and represent lengths measured over the hand. The structure of the vector is shown in Figure 3.11.

Figure 3.11: 2D feature vector data visualization. The numbers denote the vector dimensions.

In case of the **3D feature vector**, both the length and the values are different. This vector is 138 values long. The values of the distances and the widths are expressed in millimeters. The structure of the 3D feature vector is visualized in Figure 3.12.

Figure 3.12: 3D feature vector data visualization. The numbers denote the vector dimensions.

Both vectors use only the good features. From all the features available after the feature extraction, only the ones that have a good separability are used. The so-called good features are selected according to the analysis described in Section 4.1.

3.5.1 Matching methods

Many methods to compute similarity or dissimilarity of two vectors have been proposed in the past. One of the possible solutions is to use some distance measure. When using a distance measure, all the components of the feature vectors should be expressed in the same units and they should have a similar range. Otherwise, the components having the highest values would dominate the final distance and the matching would most probably not have a good performance.

In this work, two different solutions for the feature matching are available and it can be always chosen which one to apply. The first solution, probably the simplest one, is using separate matching of the 2D and 3D feature vectors with simple distance measure followed by the matching score fusion. The second approach works again with both the 2D and 3D feature vectors, however it applies metric learning in order to learn a transformation that converts the feature data into a better space for the comparison. Both are described in the following subsections.

Separate 2D and 3D feature matching using simple distance measure

In case of the separate 2D and 3D feature vectors comparison, different distance measures can be considered. The simplest and well known distance measure is the Euclidean distance[2]. There are however more sophisticated distance measures like the Hamming distance[2], the Mahalanobis distance[2], the Manhattan distance[2] and many more. For the purpose of both 2D and 3D feature vector comparison, the Mahalanobis distance is used.

The Mahalanobis distance is given by

$$d(x,y) = \sqrt{(x-y)^T S^{-1} (x-y)} \tag{3.6}$$

where x and y are the two compared feature vectors, S is the covariance matrix of the features.

Typically, the full covariance matrix S is approximated by a diagonal-only matrix. That actually degrades the Mahalanobis distance to so-called normalized Euclidean distance. The covariance matrix is in this case used to assign a weight to every component of the feature vectors according to how valuable information is carried in the particular components. The weights are computed using the analysis of the testing data described later in Section 4.1.

2D and 3D match score fusion

It is required to compare the 2D and 3D feature vectors separately. This arises from the fact that the comparison of the values stored in the 2D and 3D feature vectors gives totally different ranges of the matching scores. Thus, those scores cannot be merged together directly by just summing them up.

There are many methods for the match score fusion that have been proposed for the multimodal biometric systems in the past. In this solution, methods based on the score normalization using fixed training set were chosen. As stated in [17], in order to get a good normalization scheme, the estimates of the location and scale parameters of the match score distribution must be robust and efficient. The robustness represents the insensitivity to the presence of the outliers. The efficiency is the proximity of the obtained estimates to the optimal estimates when the distribution of the data is known.

There are many methods for the score normalization that have different properties. For this work, three different score normalization techniques were evaluated. Those are namely the Min-max normalization technique[17], Z-score normalization technique[17] and the Median and MAD normalization[17]. After the evaluation described in more detail later in Section 4.1.4, the **Median and MAD normalization** technique represented by Equation 3.7 was chosen.

$$ns_i = \frac{s_i - \text{med}}{\text{MAD}} \tag{3.7}$$

In the Equation 3.7, ns_i is the normalized score of input sample i, s_i is the original score of input sample i, med is the median value of the scores of all the training samples and MAD is so-called median absolute deviation. The equation was described more in Section 2.3.

After the score normalization process, both scores can be summed up to get the final match score. In order to do that, weighted sum expressed by the following equation

$$ms = w_{2D} \cdot ns_{2D} + w_{3D} \cdot ns_{3D} \tag{3.8}$$

where w_{2D} and w_{3D} are the weights of the 2D and 3D matching scores respectively and ns_{2D} and ns_{3D} are the normalized matching scores of the 2D and 3D matcher respectively, is used. Both scores are assigned different weights according to how well they perform on the evaluation dataset. The process of the weight estimation is further described in Section 4.1.

Metric learning based matching

Apart from the simple distance computation, metric learning performs the necessary normalization of the values internally. Therefore, even if the features do not have the same ranges of values and not even the same units, they can be directly compared by first transforming those features into a space that is better for the comparison and then computing the distance.

In order to be able to transform the features into another space, the data transformation matrix has to be estimated. This is the point where the metric learning comes into play. In order to compute the transformation matrix, a set of training samples is provided and the supervised LMNN method described earlier in Section 2.3 is used.

To compute the distance, the Equation 3.6 described in the previous section was used. In this case, the covariance matrix S represents the transformation matrix trained by using the metric learning.

The transformation matrix is trained separately for the 2D feature vector and the 3D feature vector. Thus the distance computation represented by Equation 3.6 is done twice, separately for 2D and for 3D feature vectors. The final score is then the lower of the 2D and 3D matching scores as described by Equation 3.9.

$$\mathrm{dist}\,(x,y)_{final} = \min\left(\mathrm{dist}\,(x,y)_{2D}\,, \mathrm{dist}\,(x,y)_{3D}\right) \tag{3.9}$$

As presented later in Section 4.1.6, this supervised metric learning approach generally provides much better results than the simple weighting and distance computation, and therefore, it is used by default.

Chapter 4

Experiments and results

In this chapter, experiments that were done in order to evaluate the created solution are documented and their results are presented. At the beginning the tools that were used are mentioned. Afterwards, the experiments themselves are described.

The experiments can be basically divided into two groups. The first group are experiments done using a smaller database. Those are done in order to evaluate the selected feature vectors and get some parameters that can be incorporated into the system in order to increase its performance. The second group on the other hand contains experiments done on a bigger database, which are done in order to evaluate the overall performance of the system on a more reliable set of data, while keeping the parameters obtained during the previous experiments already incorporated into the system.

This chapter is further divided into two main sections according to the pattern mentioned in the previous article. Therefore the first section provides information about how the parameters used in the system were estimated. The latter section provides details on the overall performance of the created system.

4.1 System parameters estimation

As already mentioned at the beginning of this chapter, small database is enough for the parameters estimation. In this case, small database means 25 people with ten samples for each one of them.

There are several parameters that have to be estimated in order to get a good performance of the final system. The following subsections are devoted to the description of estimation of the particular parameters.

4.1.1 Features intraclass variability testing

One of the very important things that has to be tested is the stability of the selected features. First of all, the features have to be extracted from the input images and put into the feature vectors. Those feature vectors are saved into a file that was described in Section C.

This evaluation has to be done separately for 2D and 3D feature vectors for the following simple reason. The 2D feature extraction is done using 2D images and extracted distances are in units of pixels. On the other hand, 3D feature extraction is done using the 3D point cloud that is in millimetres.

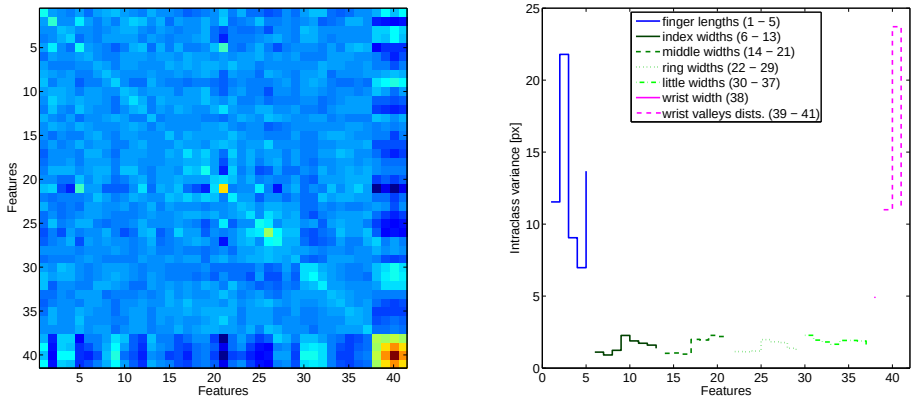
To perform the evaluation, an approach using the statistical variance of the feature vector components is chosen. The statistical variance of a set of values is computed using the following equation

$$var(x) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{4.1}$$

where x is the set of numbers and μ is the mean value of the set. It expresses how much are the values spread from the mean value of that set. Therefore the variance values of the components of the feature vector directly correspond to the stability of those components. The lower the variance for a particular component is, the more stable that component is.

The variances, however, are not obtained by computing the variance separately for each component of the vector. They are rather extracted from the covariance matrix. All the vectors are put into a matrix and then covariance matrix is computed. On the diagonal, the covariance matrix contains the variances for the feature vector components.

The covariance matrix is visualized as a 2D plot in Figure 4.1(a) for the 2D feature vector, where blue, yellow and red means low, medium and high variance respectively. The darker the color is, the higher the number is. For better visualization, the diagonal of that covariance matrix is shown separately in Figure 4.1(b). The same plots are shown in Figure 4.2(a) and 4.2(b) for the 3D feature vector.



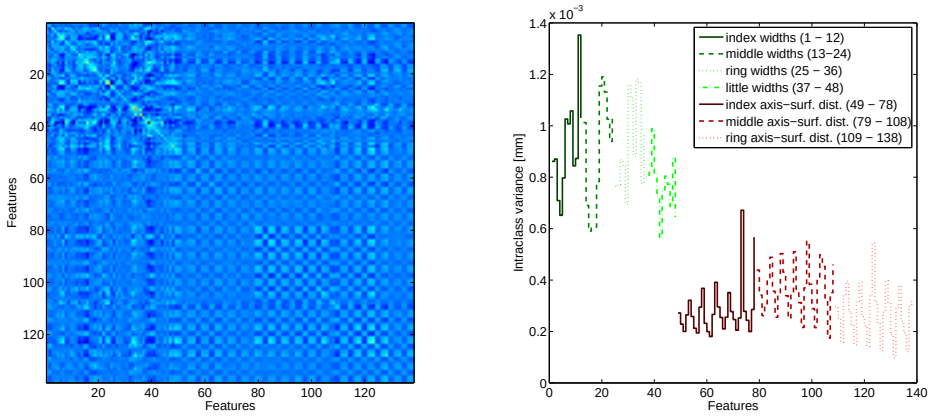
(a) (b)

Figure 4.1: Intraclass variance for 2D feature vector: (a) Covariance matrix; (b) Covariance matrix diagonal.

In case of the 2D feature vector, the first components are finger lengths, then it comes to finger widths and the end of the vector represents wrist distances. Thus it can be easily deduced from the Figure 4.1(a) that most stable features are the finger widths and the least stable ones are the wrist distances.

For the 3D feature vector, the first components are finger widths and then the distances between finger segment axis and the finger surface. Therefore the most stable components are the axis-surface distances, while the less stable ones are finger widths.

However the intraclass variability is not the only important property of the features. Before the final decision, which features are good and which are bad, was made, interclass variability was also tested. This is described in the following subsection.



(a) (b)

Figure 4.2: Intraclass variance for 3D feature vector: (a) Covariance matrix; (b) Covariance matrix diagonal.

4.1.2 Features interclass variability testing

Another important thing of the extracted features that has to be tested is how much does the feature change for input samples from different individuals. As in case of the intraclass variability testing, feature vectors are created first and saved in the file described earlier in Appendix C.

The evaluation has been done again separately for the 2D feature vectors and the 3D feature vectors for the same reason that was described in the previous section.

To measure how much the value of a particular feature changes for samples from different individuals as well as for the previous test, statistical variance was used . Also here, the statistical variance directly corresponds to the ability of the feature to differentiate between multiple individuals. However, in this case, the meaning is opposite than in the previous section. That means, the higher the variance is, the higher the value of that particular feature is.

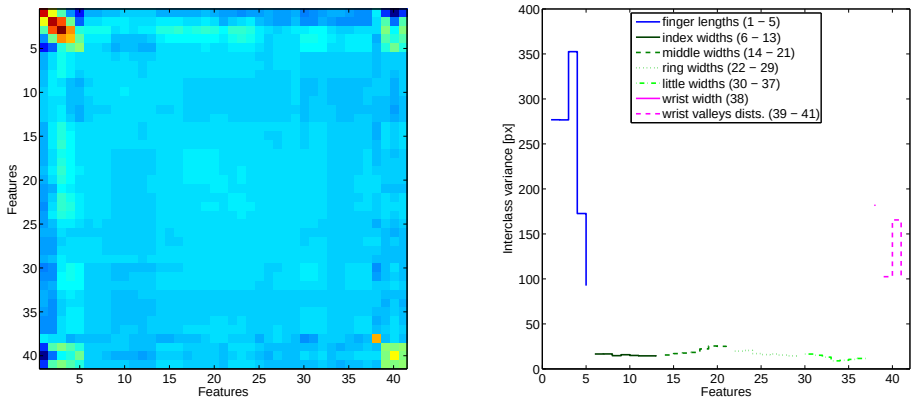
Statistical variances for each component of the feature vector are again obtained using covariance matrix by following the same approach that was described in the previous section.

For the 2D feature vector, the covariance matrix is visualized in Figure 4.3(a). The coloring of the plots is the same as in the previous section. The separated diagonal that contains the variances is shown in Figure 4.3(b). Those plots for the 3D features are shown in the two figures above, namely Figure 4.4(a) and 4.4(b).

As said already in the previous section, in case of the 2D feature vector, the first components are finger lengths, then it comes to finger widths and the end of the vector represents wrist distances. Considering what was just said, the most distinctive features can be seen in the Figure 4.3(a), in this case finger lengths and wrist distances.

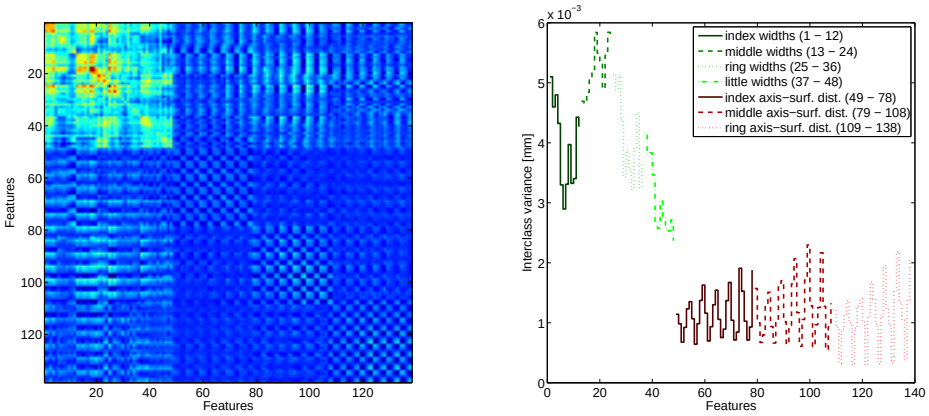
The 3D feature vector is composed from different features than the 2D one. First components are finger widths and then the axis-surface distances. As it is visualized in Figure 4.4(a), the most distinctive components are the finger widths, while the axis-surface distances are the less distinctive ones.

At this point, the intraclass variabilities as well as the interclass variabilities are known, and therefore they can be analyzed together. This is described in the following section.



(a) (b)

Figure 4.3: Interclass variance for 2D feature vector: (a) Covariance matrix; (b) Covariance matrix diagonal.



(a) (b)

Figure 4.4: Interclass variance for 3D feature vector: (a) Covariance matrix; (b) Covariance matrix diagonal.

4.1.3 Evaluation of components of feature vector

Using both the intraclass variability and the interclass variability information, which were obtained by applying the methods described in the previous two sections, the overall separability can be computed for each of the feature vector components.

Separability is a term that was in this case chosen to describe how good each particular component of the feature vector is. Two properties that are desired for all features were mentioned in the previous two sections. Those are namely low intraclass variability, which guarantees good stability of the feature, and high interclass variability, which describes the ability of the feature to differentiate between samples from different users. Taking those facts into account, the separability of a feature vector component is defined using the

following equation

$$\text{sep} = \frac{\text{var}_{interclass}}{\text{var}_{intraclass}}, \tag{4.2}$$

where $\text{var}_{intraclass}$ is the variance of a feature computed from multiple samples of the same class described in Subsection 4.1.1, $\text{var}_{interclass}$ is the variance of a feature computed from multiple samples all from different classes, as described in Subsection 4.1.2 and sep is the separability value.

Using the Equation 4.2, and knowing the separability for each component of the feature vector, the features that performed really badly were excluded so that they do not decrease the final matching score.

After the feature vector components were filtered and only the good features left, a weight value was assigned to each feature according to how high separability it has. The weight corresponds to the separability value. In order to keep the weights around 1.0, the separability for each feature is divided by the mean separability computed for all the features. Therefore, the weights are computed simply using the equation

$$\text{weight} = \frac{\text{sep}}{\mu_{sep}} \tag{4.3}$$

where sep describes the separability value for a particular feature and μ_{sep} is the mean separability computed from separabilities of all the features.

The following Table 4.1 shows some interesting examples of the intraclass variabilities, the interclass variabilities and the resulting weights. It can be seen that the finger lengths have good ability to distinguish between different classes however they are also very unstable. On the other hand, finger widths does not have such a good ability to distinguish between different classes but on the other hand they are very stable. Therefore, all the features shown below in the table provide valuable information and they are only weighted according to how much information they carry.

Table 4.1: Examples of variance evaluation results and weight computation results.

Value	index f.	middle f.	f. width 1	f. width 2	wrist 1	wrist 2
$var_{interclass}$	398.80	283.77	9.14	9.17	172.82	218.56
$var_{intraclass}$	16.99	11.43	1.96	1.46	16.30	18.27
$weight$	2.34	2.48	0.46	0.62	1.05	1.19

4.1.4 Biometric fusion using score normalization

As already explained in Section 3.5.1, the fusion is done using score normalization approach. In order to obtain high performance, several methods were chosen and their results were compared to find the most suitable one.

All the methods are basically based on estimation of some parameters that are then used for the normalization of the matching score. Those parameters are by default never known and therefore a training set of matching scores, which they can be trained on, has to be available. This set should contain both genuine and impostor scores and it should represent the possible distribution of the matching scores as well as possible.

For the purpose of this evaluation, the small database was used. This database, as already said earlier, contains 25 users with ten samples per each user. From those ten

samples, first four samples are used to create the user template feature vector and the remaining six are used for the testing. Thus there are seven feature vectors for each user in the end. In order to obtain as many scores as possible, both genuine and impostor, each testing feature vector is compared with the template feature vectors from all users. This way there are in total 3,675 scores; where 147 scores are genuine and 3,528 are impostor.

The estimation of the parameters has to be done separately for 2D and 3D feature vectors since they are not directly comparable, which is the reason why the whole normalization process is done.

Min-max normalization

The method called min-max normalization relies on the knowledge of the s_{min} and s_{max} parameters, which represent the minimal and the maximal score obtained from a set of matching scores respectively.

The set of scores described at the beginning of this section was used. All the scores are saved as one vector and therefore the parameters were computed using very basic Matlab functions as

$$s_{min} = \min(\text{scores}) \tag{4.4}$$

$$s_{max} = \max(\text{scores}) \tag{4.5}$$

where *scores* is a set of matching scores.

Table 4.2 shows the results of the recognition using this method evaluated on the small database of 25 people. As you can see from the table, for the small database, this method has shown the worst performance of all three tested methods. Moreover, the min-max method is not even robust.

Z-score normalization

Another solution is to use the so called Z-score normalization. In this case, normalization is based on two parameters, namely μ that describe the mean value of the matching score set and σ, which represents the statistical standard deviation of the matching scores.

Again, the set of scores is saved as one vector and can be easily processed using Matlab to get the desired parameters using the following functions

$$\mu = \text{mean}(\text{scores}) \tag{4.6}$$

$$\sigma = \text{std}(\text{scores}) \tag{4.7}$$

where *scores* is a set of matching scores, μ is the mean score value and σ represents the standard deviation of the score values.

The results of recognition using Z-score normalization are shown in Table 4.2. It can be seen that the performance is better in comparison to the min-max method. However the problem is that this method does not guarantee the same numerical range for the normalized scores of the different systems. Moreover, the distribution of the set of scores is preserved only if it is Gaussian. As well as Min-max normalization, the method is efficient, but not robust.

Median and MAD normalization

Last of the methods evaluated in this text is the Median and MAD normalization. This technique is based on computation of *scores*, which stands for Median Absolute Deviation, and estimation of *scores* parameter, which is the median of the set of scores. Those two parameters were obtained in Matlab using the following two lines

$$\text{med} = \text{median}(\text{scores}) \tag{4.8}$$

$$\text{MAD} = \text{median}(\text{abs}(\text{scores} - \text{med})) \tag{4.9}$$

where *scores* is a set of matching scores. Results of the recognition using this approach are shown in Table 4.2. This method is insensitive to the presence of outliers, however in comparison to the Z-score normalization, its efficiency is low. Also, the distribution of the data has to be Gaussian, otherwise the *scores* and *scores* parameters used by this method represent the properties of the dataset very poorly.

But still, as you can see from the Table 4.2, this method gives the best results on the testing dataset in overall and therefore it is the one being used in this solution.

Table 4.2: Recognition performance using different score normalization methods. Min score and max score are minimum and maximum matching scores respectively. The *Error* is the number of wrongly recognized samples on the test dataset using simply the lowest distance based classification and *EER* is the equal error rate computed from the results on the testing dataset.

Method	min score	max score	*Error*	*EER*
Min-max	0.00	6.74	1.36%	2.69%
Z-score	-3.28	33.53	1.36%	2.72%
Median and MAD	-5.29	57.50	0.68%	2.69%

4.1.5 Transformation by applying metric learning

From what was shown in the previous section, one can see that using score normalization methods provides acceptable performance of the resulting system. Using the Median and MAD score normalization method, the results are quite good, however errorneous of the system on the testing set is not the only criteria, it is also very important that the genuine and impostor scores are well separable so that a good decision threshold can be chosen. As it is presented later in Subsection 4.1.6, the separability using the score normalization methods is not always very good.

Therefore, there were attempts to come up with a different solution to the fusion of 2D and 3D biometric recognition. The strive was to get better results and thus a supervised learning method for distance computation was chosen. Namely, it is the LMNN metric learning method, which was explained earlier in Section 2.3.

The approach is based on using a transformation matrix that has to be trained on a training dataset. The training dataset was chosen as a subset of the small database and it contains only 12 users. For the training, four samples of each user from the subset are used, therefore in total it is 48 samples for the training purposes. The learning was done using

existing code from prof. Weinberger[20] for LMNN metric learning in Matlab as already mentioned in Section C.

To be able to compare the LMNN metric learning approach to some other approach that does transformation of the data, PCA method was used. The comparison is shown below in Table 4.3.

The evaluation is done again on the whole small database of 25 people, separately for 2D and 3D biometric features. However, in this case, there is no need to weight the components of the feature vectors since using this method, the weighting is done internally by the trained transformation.

In the real world applications, it is however very important to be able to enroll new users into the database continuously throughout the system lifetime. In such cases, it is not possible to train the transformation matrix on all the users in the database. The transformation matrix is trained on provided training dataset that does not contain all the users since they can be enrolled anytime in the future. Therefore the transformation matrix is used also with the new unknown users, which it was not trained on. As already said, the transformation was trained only on a small subset of the testing data. Therefore the results shown in Table 4.3 also prove that even if the transformation matrix is not trained using all the data, it performs well and therefore it can be used also in the systems, where the database is being extended continuously during the system lifetime.

Table 4.3: Recognition performance using metric learning approach. Min score and max score are minimum and maximum genuine matching scores respectively. The *Error* is the number of wrongly recognized samples on the test dataset using simply the lowest distance based classification and EER is the equal error rate computed from the results on the testing dataset.

Method	Features	min score	max score	*Error*	EER
PCA	2D + 3D	0.03	5.55	5.47%	5.13%
LMNN	2D	0.20	169.54	2.05%	2.06%
LMNN	3D	1.14	172.31	4.10%	4.70%
LMNN	2D + 3D	0.20	154.14	0.68%	2.04%

4.1.6 Decision threshold estimation

Decision threshold is one of the most important things when it comes to biometric system. It defines how the system evaluates the computed matching scores that it gets from the matcher. To have a good performance of the system, the threshold has to be set appropriately according to the current requirements of the application. The requirements may change with different applications and therefore there is not a single threshold that would always be the best one.

In case of this work, the goal was to estimate a decision threshold that balances the FAR and FRR values, which is the most commonly used criteria. The threshold is usually being estimated using the ROC or DET curves that were described in Section 2.4. The following two subsections describe the score estimation in more detail.

Score distribution histograms

Before actually plotting the ROC curves, the obtained matching scores were visualized using a histogram. For both genuine and impostor scores, a histogram of 100 bins was created to visualize the score distribution. The histograms of genuine and impostor scores were in the end put together to emphasize the overlapping of the genuine and impostor score distributions. By analysis of this overlapping, the proper decision threshold can be found.

The histograms created for each of the tested methods are shown in the Figures 4.7 and 4.6. First of the figures shows the histograms using metric learning approach. The second figure shows the histograms for the different score normalization methods. In each histogram, blue colored distribution are the genuine scores and the red colored distribution are the impostor scores.

With a closer look at the histograms in Figure 4.6 and 4.7, one can clearly see that the score distributions of the genuine and impostor scores have a big overlap and therefore selecting a decision threshold that would guarantee perfect performance in means of number of mistakes in the recognition process is not very well possible.

Apart from that, by analyzing histograms in Figure 4.6, it is obvious that metric learning approach improves the separability of the score distributions . The results will never be ideal. That means the scores cannot be separated without making any mistakes. However, the decision threshold can be still set so that the performance of the system is very high.

ROC curve based threshold selection

Usually it is not very suitable to estimate the decision threshold from the score distribution histogram. The histogram is a good visualization to see the overlapping of the distributions, however in order to estimate the precise decision threshold, it is much better to plot the ROC curve and get the threshold value by its analysis.

As already said in Section 2.4, in order to balance the FAR and FRR values, the aim is to select such a decision threshold that corresponds to the top-left most point on the ROC curve. However, the manual analysis of the ROC curve is not the best and not even the easiest way to get the desired threshold value and therefore Matlab was used and the threshold was estimated automatically. Extraction of such a threshold can be easily implemented in Matlab. Plots of the generated ROC curves are shown in the Figure 4.5 above. The coloring is explained in the legend attached to each graph. As you can see from the ROC curves, the metric learning approach performs better than the score normalization methods in case the combination of 2D and 3D features is used.

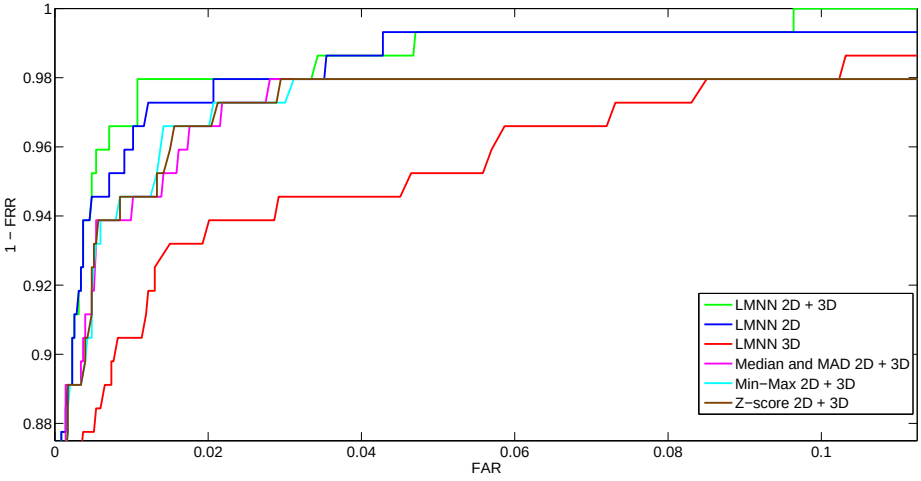


Figure 4.5: ROC curves of the system using different fusion methods.

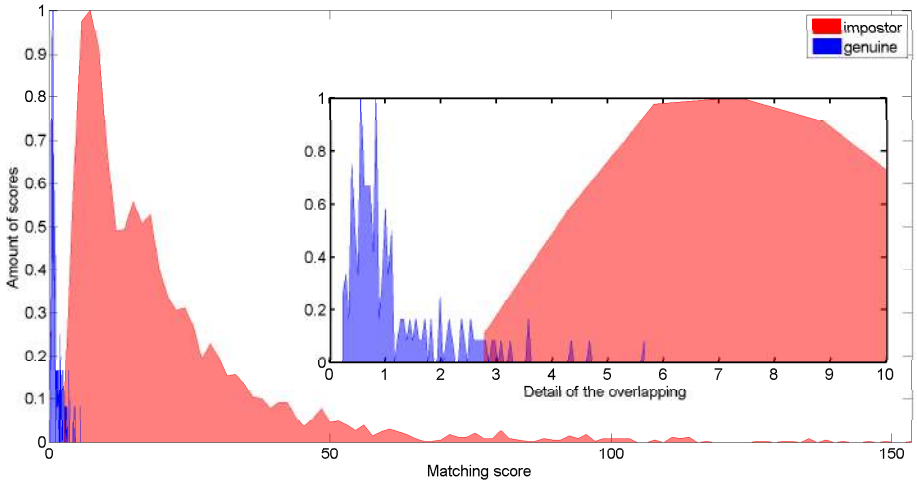
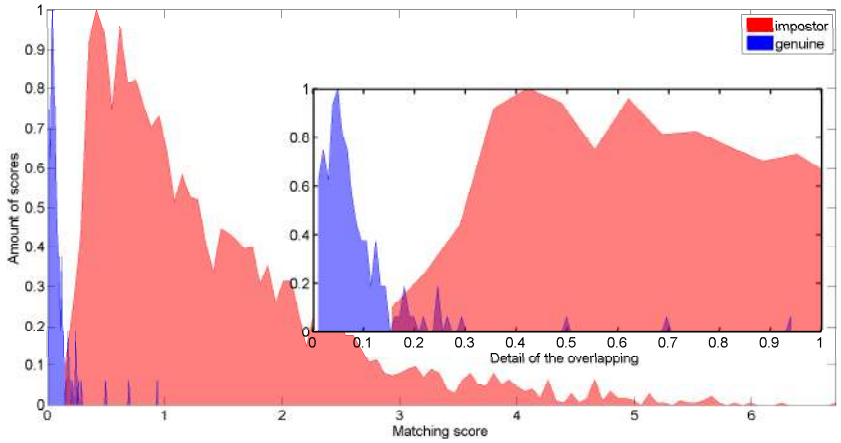
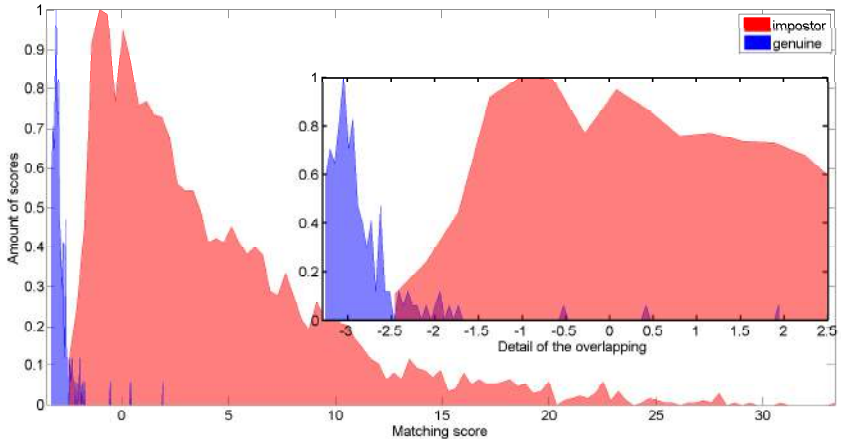


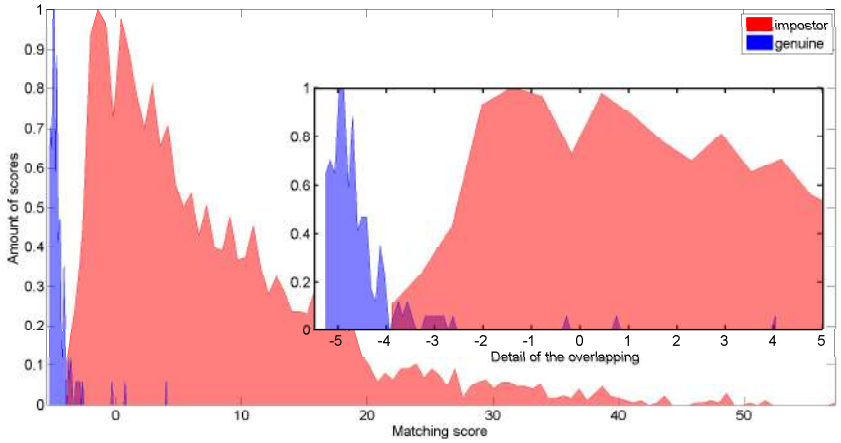
Figure 4.6: Histograms of genuine and impostor distribution overlapping for metric learning approach.



(a)



(b)



(c)

Figure 4.7: Histograms of genuine and impostor distribution overlapping for different score normalization methods (a) Min-max normalization; (b) Z-score normalization; (c) Median and MAD normalization.

The solution will never be perfect, i.e. the EER will never be 0. In such a case that perfect results might even indicate that the system is most probably overfitted on the training data set. To make sure this is avoided, the final transformation matrix is created using metric learning in combination with cross validation on the small database, i.e. repeating the training process by selection different samples for the training for each user from the small database.

It might seem from Table 4.3 and Figure 4.5 that the combination of 2D + 3D does not improve the overall performance as much as expected. Therefore more tests were done in order to prove that it is not so and that the 3D features provide important information that can improve the performance significantly. This test is described in the following subsection.

4.1.7 Testing on the problematic users subset

Even if the overall performance of the system is good, its sensitivity to the problems during the feature extraction can be high and performance in such particular cases might get much worse.

Using multiple feature vectors can decrease the impact of those problems. In this solution, both 2D and 3D geometric information are extracted from the hand surface and two feature vectors are created. Matching scores obtained from the separate comparison of the 2D and 3D feature vectors are in the end combined as described earlier in Section 3.5.1. This fusion of the 2D and 3D geometry recognition helps in case there are so-called problematic users in the dataset, i.e. users whose hand are more difficult to process.

For the purpose of performance evaluation in such a special case, new dataset was selected as a subset of the big database of 100 users. This dataset contains mainly users whose samples are more difficult to process in means of feature extraction. The special dataset contains 17 users with 173 samples in total. The first four samples of each user are used for the creation of the template, the other samples are used for the testing.

Examples of the typical problems during the feature extraction process are shown in Figure 4.8, 4.9 and 4.10.

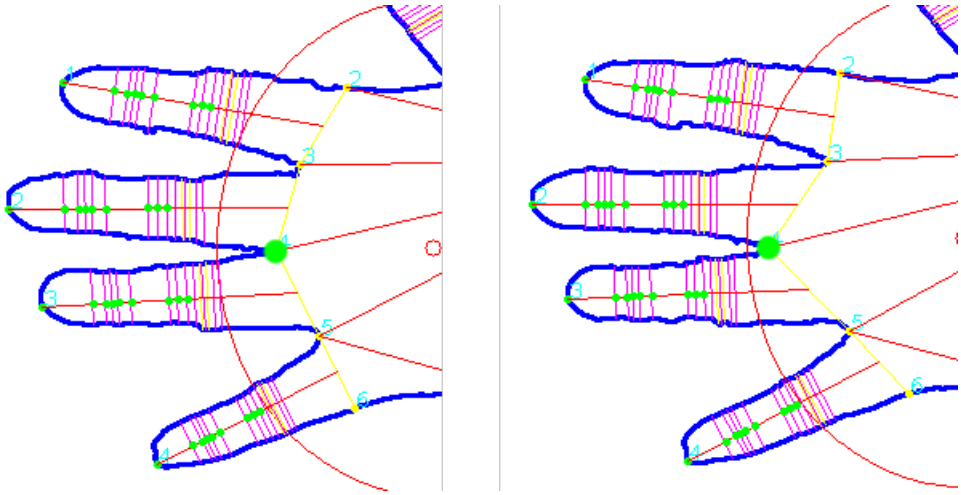


Figure 4.8: Errorneous finger valley extraction due to the middle and ring finger being too close together in combination with a noise during the acquisition.

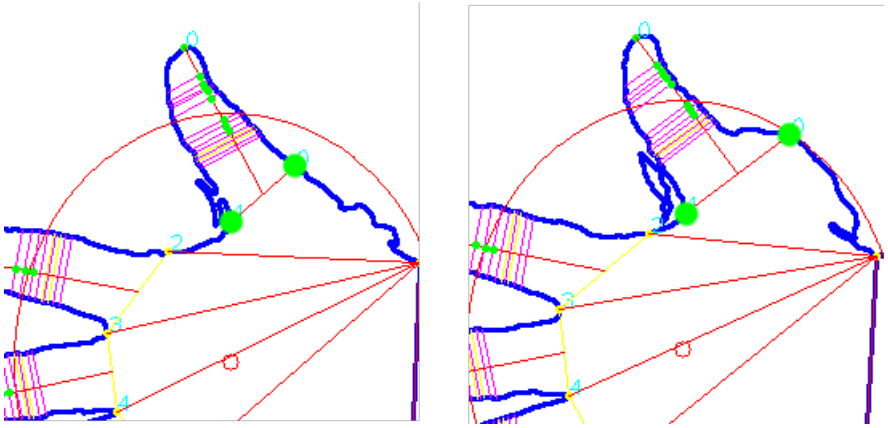


Figure 4.9: Errorneous finger valley distance computation due to the hand being too close to some surface under it. That caused problems with foreground and background separation which has this particular effect.

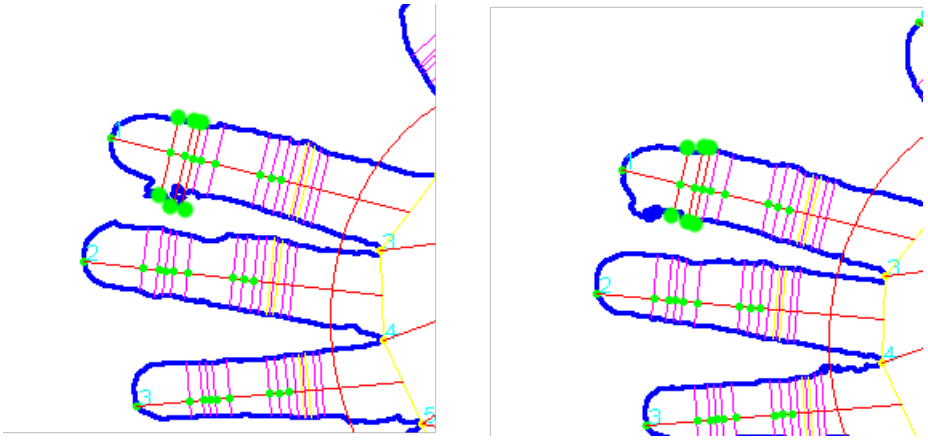


Figure 4.10: Errorneous finger widths computation due to the noise during the image acquisition.

The performance was tested again for all the methods presented earlier. The results are shown in Figure 4.11 below.

As you can clearly see from the ROC curves in Figure 4.11 and Table 4.4, the score normalization methods are the average on this dataset. On the other hand, the LMNN metric learning approach using only 2D information has very bad performance, which is caused by the problems during the feature extraction. The standalone 2D matching is very sensitive to the problems during the feature detection process. However, this problem is balanced by the standalone 3D feature matching, which performs much better in case of problematic users dataset. As an outcome, the fusion of 2D and 3D information using the LMNN metric learning gives very good final performance of the system.

Table 4.4: Performance of the recognition on the problematic users dataset. The EER is the equal error rate computed from the results on the testing dataset.

Method	Features	EER
Z-score	2D + 3D	3.83%
Median and MAD	2D + 3D	3.77%
Min-Max	2D + 3D	4.00%
LMNN	2D	5.36%
LMNN	3D	3.77%
LMNN	2D + 3D	2.83%

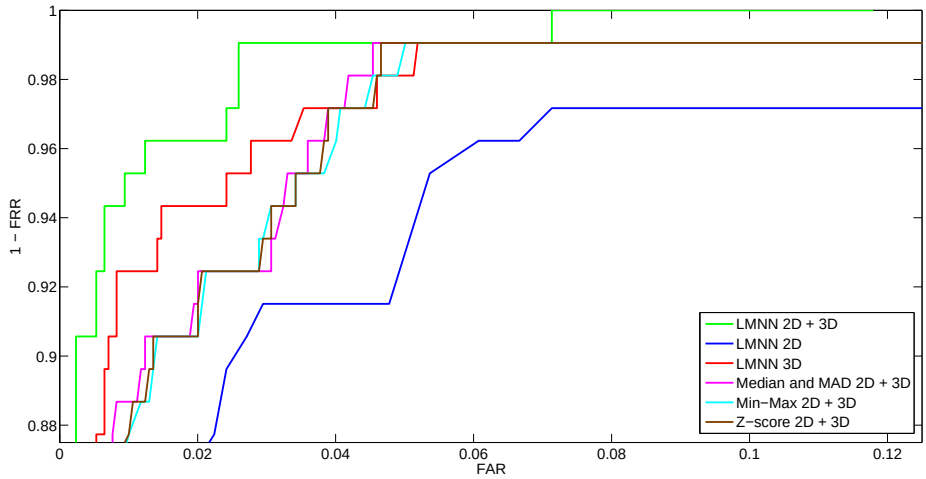


Figure 4.11: ROC curves of the system using different fusion methods on the problematic users dataset.

The point is, that in this case, using the 3D features improved the overall performance a lot and therefore it demonstrates the importance of using not only 2D, but also the 3D information.

4.2 Performance evaluation

After all the parameters were estimated and set, performance evaluation of the system was done. In order to do reliable performance evaluation, it is required to have a bigger set of testing data, i.e. database of at least approximately 100 users.

However, the small database used mainly in the previous section was used also for one performance test. This test is described in the Subsection 4.2.2.

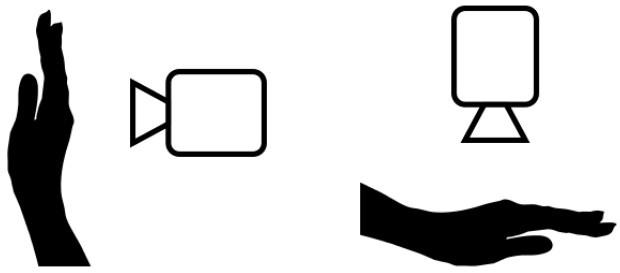
First of all, a short summary of the experience with the data acquisition is presented in the following subsection.

4.2.1 Data acquisition experience

A lot of data were collected in order to create a bigger testing dataset of approximately 100 users. During this process, many different individuals interacted with the system and their experience with data acquisition could have been observed. From the observations, the following important rules that should be satisfied by the data acquisition process were imposed:

- the user should be informed how to interact with the device;
- the hand should be presented in a way that the position of the hand feels natural for the user;
- the device should be mounted in the height between chest and abdomen of the user.

Since the system described in this text is completely new, the first rule is quite obvious. In general, new users have never seen a device like that in the past and they have no idea how to interact with it. It is therefore needed to show them how to put the hand in front of the sensor and what information to follow on the screen. However, if the users are informed, it was observed that the device is very easy to use and in general the users did not have any problems presenting their hands.



(a) (b)

Figure 4.12: Possible setups for the hand acquisition: (a) Hand vertical placement (poorly accepted by the users); (b) Hand horizontal placement (well accepted by the users).

The second rule was raised in the early phase of the development. At first, the hand was supposed to be presented vertically, with the fingers pointing up and hand palm facing the user as shown in Figure 4.12(a). However, when non-experienced people were asked to place their hands in front of the camera properly, it took them very long time to do so, and

moreover, for some of them it was nearly impossible. Thus, a different approach was carried out. The camera faces the ground and the hand is supposed to be placed horizontally below the camera with fingers pointing away from the user and the hand palm facing the ground (see Figure 4.12(b)). In this case, having the camera placed in the proper height, it was observed that the usage is very simple and feels natural for the users.

The last rule is directly derived from the previous article and the need to have the camera placed in the proper height. If the camera is placed too low, the users tend to lean the whole hand down and that creates a problem with the correct hand positioning. Similar problem arises when the camera is placed too high, the users tend to lean the whole hand up and that again creates a problem with the correct hand positioning. From multiple observations of users interacting with the camera, it was observed that the proper height is approximately between the abdomen and chest of a user.

4.2.2 Observing hand under various transformations

In this solution, hand is captured in the air, without being placed on some surface. As presented in Section 3.2, there are some constraints for the hand positioning. There is even so-called positioning loop that leads the user to the correct positioning of their hand. However in order to be able to use the system without serious issues, these constraints cannot be very limiting. Therefore it can happen that the hand is sometimes not placed really parallely with the camera, but a bit rotated to one of the sides.

(a) (b) (c) (d)

Figure 4.13: Hand observed under different transformations: (a) Bent fingers; (b) Leant down; (c) Leant right; (d) Leant left.

In order to evaluate behavior of the system under such circumstances, one individual was captured with multiple different rotations of the hand in front of the camera. This individual was also previously captured for the small database and therefore the system has been trained to recognize his hand from the images where it is placed properly. Then, the images with different rotations are presented to the system and the recognition is performed.

As already said, there are several different rotations of the hand. All the introduced rotations are listed below. Examples of all the transformations are also visualized in Figure 4.13.

- **bent fingers** - hand is placed more or less parallely, but the fingers are bent;
- **leant down** - hand is leant in a way that the fingers are pointing a bit down;

- **leant left** - hand is leant to the left;
- **leant right** - hand is leant to the right.

Particularly, two levels of transformations were experimented with. The first group are transformations, which are still in tolerance of the introduced positioning system or which are on the edge of the tolerance. The second group of transformations are the same types, however this time the hand is transformed more and the positioning constraints are not satisfied anymore.

The first group of transformations was captured in order to confirm that the positioning tolerance is reasonable, i.e. it is still possible to position the hand easily but also the small transformations that are allowed are handled properly by the system.

The second group of transformations was obtained so that it can be observed what would happen in case the positioning tolerance would be increased.

The captured input images were added into the small database and recognition was performed. The histogram in Figure 4.14 shows the distribution of the genuine and impostor scores for the smaller transformations. Having a closer look at the histogram, one may see that there are some genuine scores far above the decision threshold. However, in this case, the decision threshold can be still set so that the genuine and impostor scores are separated in an acceptable way. One way or the other, the overall performance of the system is significantly decreased.

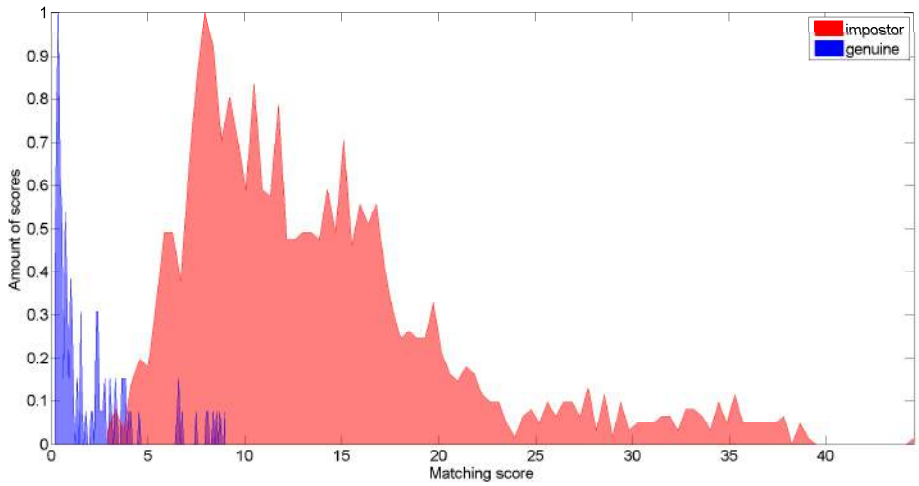


Figure 4.14: Histograms of genuine and impostor distribution overlapping for the smaller transformations dataset.

Histogram in Figure 4.15 shows the genuine and impostor score distributions in case the samples with bigger transformations are inserted into the database. By observing the histogram, it is clear that with bigger transformations of the hand in the input images, the distributions are not so separable anymore and performance of the system is decreased dramatically.

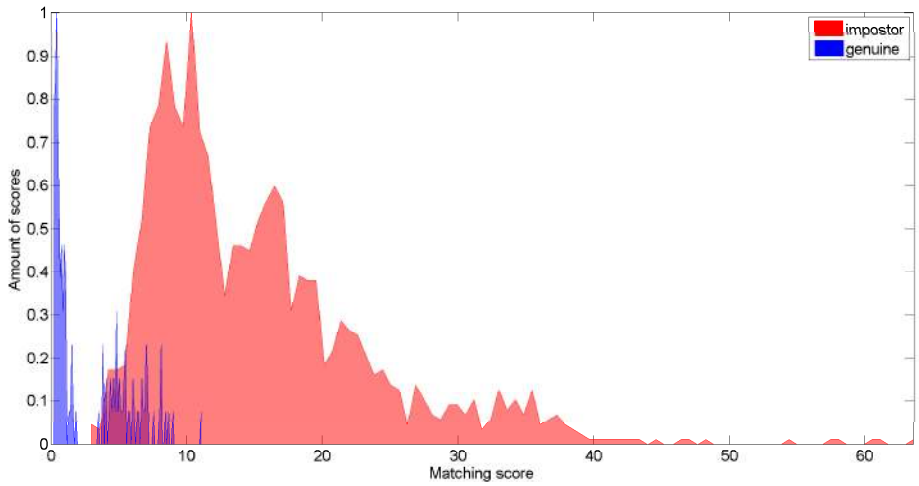


Figure 4.15: Histograms of genuine and impostor distribution overlapping for the bigger transformations dataset.

The ROC curves in Figure 4.16 show how the performance of the system is decreased from smaller to the bigger transformations. It is clearly visible that the system has serious issues when it comes to processing of samples that contain bigger transformations of the hand.

For the smaller transformations dataset, the EER is 8.47%, for the bigger transformations, the EER is 9.38%. It may seem that the difference is not so big. This is however not true. It is obvious from the ROC curve that by trying to decrease the FAR for the bigger transformations dataset, the FRR would be increased dramatically.

To show which transformations are the most problematic, a few samples were chosen from the smaller transformations dataset and they are shown in Table 4.5. As you can see, the most problematic case is when the hand is leant left. On the other hand, when the fingers are bent just a bit, the system can deal with it as well as with the cases of leaning down or leaning right.

It can be clearly observed that the transformations of the hand make the recognition process very unstable. In case of the bigger transformations dataset, the recognition often does not give a correct result. However, even if the result is correct, in many cases the matching score is too high and it would probably be over the decision threshold. This means that the system would fail to recognize the sample in the end. In case of smaller transformations dataset, the system is able to deal with the changes quite well, but the scores are getting closer to the decision threshold, which indicates that the system is becoming unsure about its decisions.

This experiment has shown that in the current implementation, it is very important to pay attention to the positioning of the hand in order to keep the good performance of the system. Therefore, the positioning phase and the process called positioning loop introduced in Section B.2 are very important.

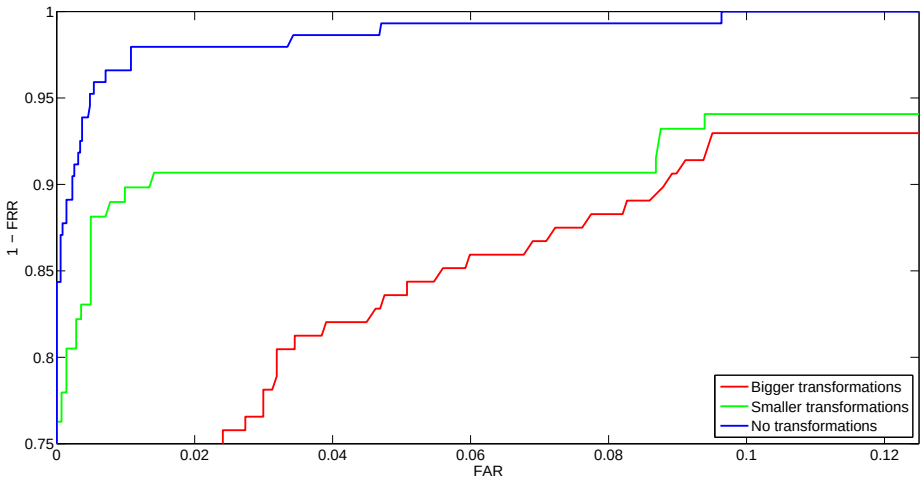


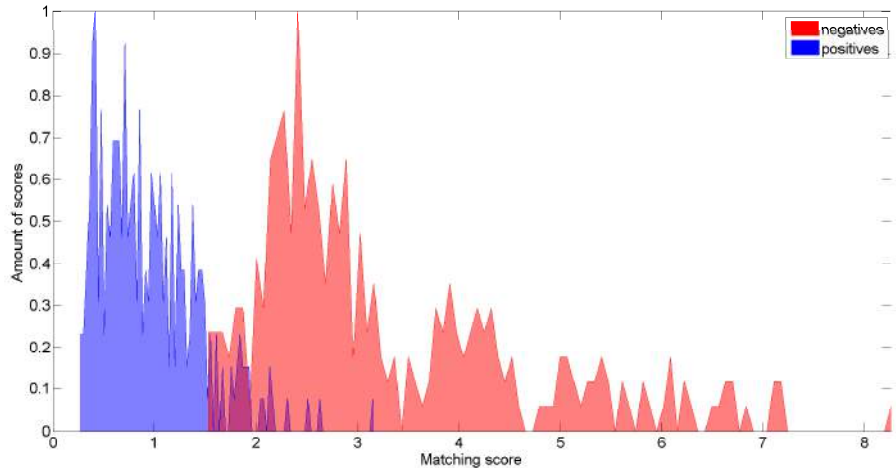
Figure 4.16: ROC curve obtained from the datasets that contain smaller and bigger transformations.

Table 4.5: Results of the recognition with input samples under different transformations. All the input samples are from class 1. All the results of the recognition are shown in the format „class (score)“.

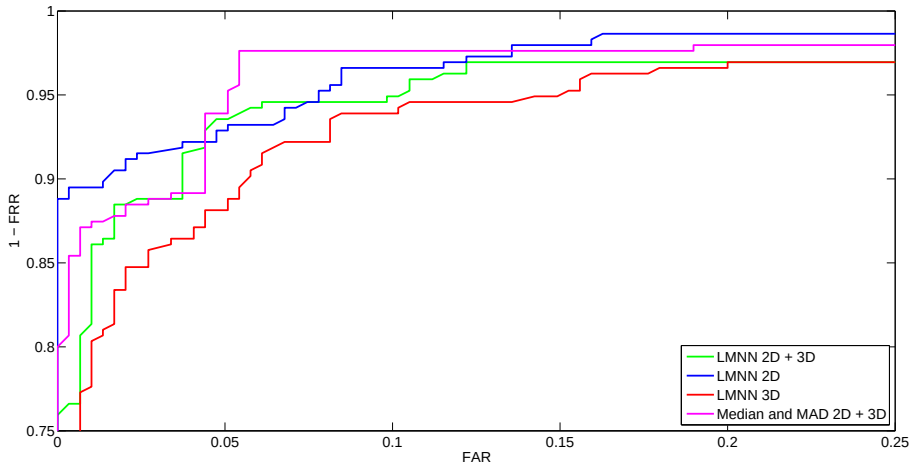
Transformation	sample 1	sample 2	sample 3	sample 4
bent fingers	1 (6.56)	1 (2.45)	1 (2.48)	1 (2.73)
leant left	1 (6.56)	10 (7.47)	10 (8.09)	1 (9.02)
leant right	1 (8.72)	1 (2.56)	1 (2.34)	1 (2.77)

4.2.3 Classification according to gender

To experiment with general capabilities of the system, a special test was performed. The aim of the test was to recognize the gender of an individual according to the geometry of his/her hand.



(a)



(b)

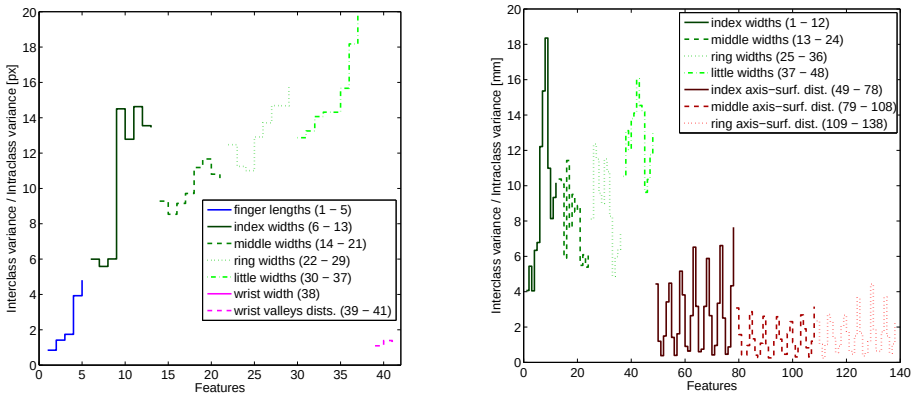
Figure 4.17: Results of the classification according to the gender (a) Histogram of the positive and negative score distributions; (b) ROC curves.

A part of the bigger testing database that was used for evaluation of the system performance contains also information about the gender of the captured individual. This set is 52 users out of the total size of 100.

Naturally, the gender can be either male of female, nothing else, and therefore in this case the problem degrades to only two class classification.

To observe the differences between the hand geometry of the male and female individ-

uals, variability was computed in order to show which features differ the most between a male and a female individual. The computed variabilities for each feature are visualized in Figure 4.18.



(a) (b)

Figure 4.18: Variabilities between male and female classes: (a) 2D features variabilities; (b) 3D features variabilities.

The classification was again done using all the available approaches, which means either score normalization methods (only the best performing one - Median and MAD normalization) or the metric learning approach. The male and female classes are quite well separable as the histogram in Figure 4.17(a) demonstrates. Performance of the classification according to the gender is represented by the ROC curve in Figure 4.17(b).

As you can see, paradoxly, in case of classification according to the gender, the LMNN approach performs worse than the Median and MAD score normalization approach, which is the best available for the classification according to the gender.

4.2.4 Testing with bigger database

The final performance evaluation on the bigger database is again done for both available approaches, i.e. the score normalization methods and metric learning.

In case the score normalization methods are used, the performance was expected to be lower than using the metric learning. This fact was proved and it is demonstrated in the following subsection. Therefore the score normalization approaches were used only for the verification purposes.

On the other hand, using metric learning, the performance was expected to be very good. The results are demonstrated in the Subsection 4.2.4 and they show that the metric learning approach is sufficiently good to be used for both identification and verification purposes on the database of approximately 100 users.

Weighting of features using Mahalanobis distance

As already said, the testing was done using also the score normalization methods. However, apart from the previous evaluation, histogram of only the best performing method from the previous evaluation is shown.

The previous tests have proven that the best performing out of the three evaluated score normalization methods was the Median and MAD normalization method. Thus, this method's histogram was chosen and visualized in Figure 4.19.

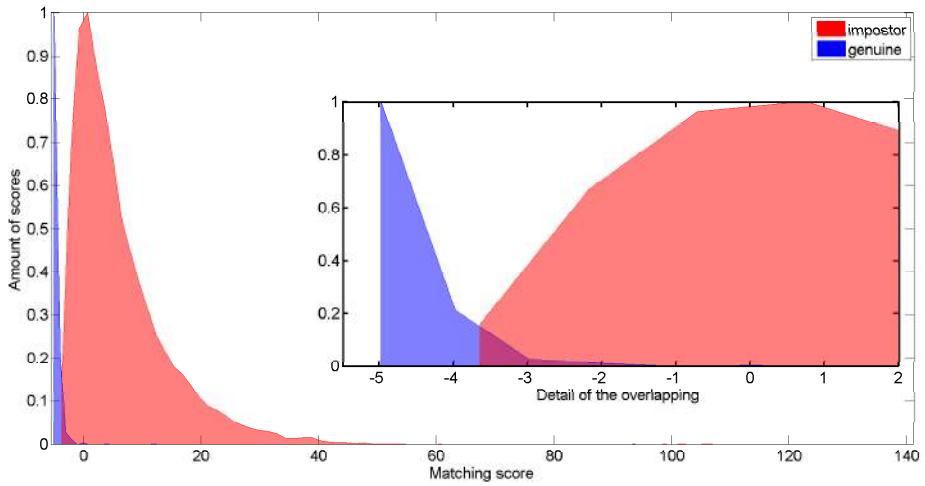


Figure 4.19: Histogram of the genuine and impostor matching score distribution obtained from the evaluation on the big database using the Median and MAD score normalization method.

The ROC curves of all the three score normalization methods, representing their performance can be seen in Figure 4.21. The Table 4.6 at the end of this section shows FAR and FRR values and also their decision threshold setups.

It can be easily seen from the Table 4.6 that for a bigger dataset, the score normalization approaches can still provide satisfying results. Having the EER value about 2.2% makes the system usable for the real world applications. However, the evaluation of the score normalization approaches was added just for completeness and the bad performance was

Table 4.6: Results of the recognition on the big dataset using all three score normalization approaches.

Normalization method	Features	EER
Min-max	2D + 3D	2.11%
Median and MAD	2D + 3D	2.05%
Z-score	2D + 3D	2.02%

expected. This was not proved in the end. Even though they perform worse than LMNN metric learning approach in general, they still perform considerably well. This fact was noticed already during experiments on the small dataset. However, in order to improve the performance of the system, another approach that provides much better results was carried out. This approach is evaluated in the next subsection.

Metric learning approach

As already said, the score normalization methods were not expected to be good enough to handle bigger datasets and therefore another approach was carried out. It is the metric learning based approach described earlier in Section 3.5.

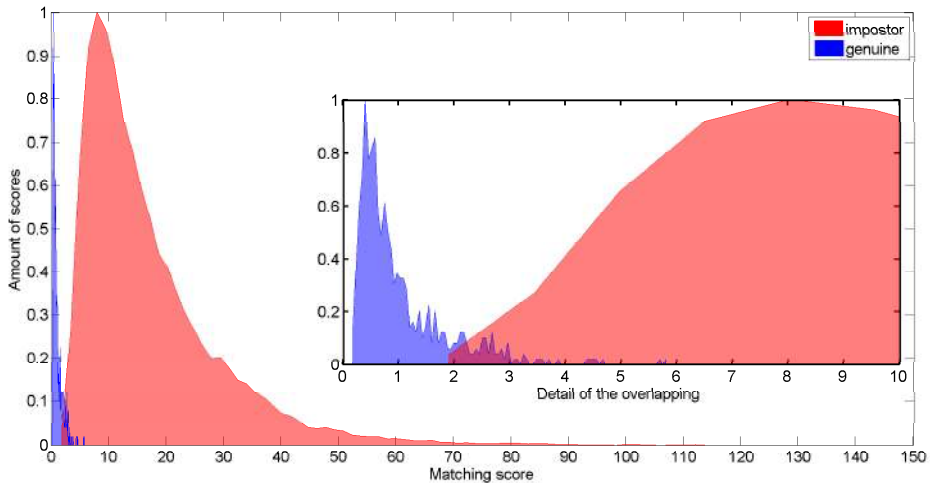


Figure 4.20: Histogram of the genuine and impostor matching score distribution obtained from the evaluation on the big database using the LMNN metric learning.

Using a training set that contains only a few users from the whole dataset, it was proved that it is sufficient to train the transformation matrix on a small subset of the whole dataset and then do the recognition on a much bigger dataset with a lot of unknown users that the transformation matrix has never been trained for. This way, the metric learning approach is able to handle bigger dataset without the need to retrain the transformation matrix every time the dataset changes.

The distribution of the genuine and impostor matching scores obtained from the eval-

uation on the bigger dataset can be seen in Figure 4.20. From the overlapping of the histograms, it is clear that the distributions are not well separable anymore. However, a decision threshold can be still chosen so that the EER is reasonably low.

The overall performance of the system is again visualized using the ROC curve in Figure 4.21. The ROC curve shows that the system is performing very well.

The EER values and the corresponding decision threshold are then shown in Table 4.7. Those results are shown for standalone 2D recognition, standalone 3D recognition and also for the fusion of both 2D and 3D.

Table 4.7: Results of the recognition on the big dataset using LMNN metric learning approach.

Features	EER
2D	2.45%
3D	2.68%
2D + 3D	1.42%

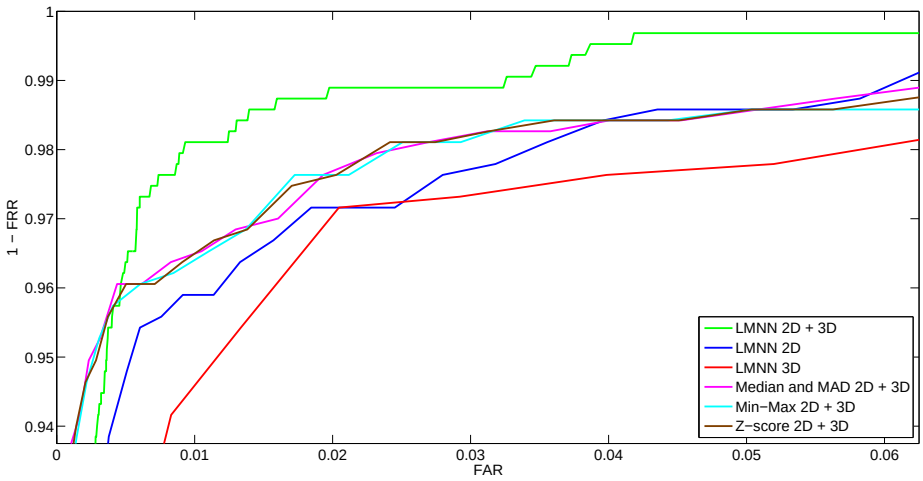


Figure 4.21: ROC curves of the system using six different matching methods.

As you can see in Table 4.7, the fusion of 2D and 3D biometric features provides the best results. The EER value for system that uses only 2D hand geometry is approximately 2.6%. System that uses only 3D hand geometry performs a bit worse, the EER is approximately 2.9%. However, by combining the 2D and the 3D hand geometry, the performance of the system is increased by approximately 45% and the resulting EER is approximately 1.42%. This confirms that what was proved earlier in Section 4.1.7 applies even for the bigger dataset (using also 3D information increases the final performance significantly).

In this section, it was also proved that the LMNN metric learning approach further improves the already good results obtained by using the score normalization methods. By examining the Figure 4.19 and 4.20, you can notice that using LMNN metric learning

approach, the gap between the distributions increases a bit which shows that the separation ability using LMNN metric learning approach is really better.

4.2.5 Comparison to the existing research

To compare the performance of the system to the existing research that had been done in the past, a list of different proposed methods mentioned in [9] from 2009 was taken. This list contains different methods that process either 2D, 3D or both 2D and 3D information that can be obtained from the hand surface.

The Table 4.8 inspired by [9] lists all the approaches that were compared with the one presented in this text. The first two approaches described further in [8] and [7] consider only 2D geometric information and they require the user to put the hand on a specific surface. The latter four approaches explained in [21], [15], [13] and [9] use both 2D and 3D geometric information and the last approach does not even impose any special requirements regarding the hand placement. However all the mentioned approaches either do not use the 3D information at all or they consider an expensive very precise 3D camera to be used.

Before consulting the actual results, it has to be mentioned that such a comparison cannot be taken as the direct reliable comparison of the methods listed in the table below. This is due to the fact that the results for each method were obtained using different datasets with different sizes. Therefore, this comparison is just approximate and it cannot be taken officially since it might be a bit misleading.

Table 4.8: Comparison of the existing approaches to the novel approach proposed in this article.

Author	Type	No. of Templates	FAR	FRR	Database size
Jain and Duta[7]	2D	1 - 14	2%	3.5%	53
Jain et al.[8]	2D	1 (Avg.)	2%	15%	50
Woodard and Flynn[21]	2D + 3D	1 (Avg.)	5.5%	5.5%	177
Malassiotis et al.[15]	2D + 3D	4	3.6%	3.6%	73
Kumar et al.[13]	2D + 3D	5	5.3%	8.2%	100
Kanhangad et al.[9]	2D + 3D	5	2.6%	2.6%	177
Our solution	2D + 3D	1 (Avg.)	1.42%	1.42%	100

The solution presented in this text tries to combine the best out of the mentioned approaches, therefore it does not require the user to put the hand on some surface and it uses both 2D and 3D geometric information. Moreover, this solution uses mainstream 3D camera that should be available for very affordable price later this year.

The actual comparison of the performances of the different solutions is summarized by the Table 4.8 using the FAR and FRR values.

As it can be easily observed in Table 4.8, the approach proposed in this text performs comparably well, in most cases even better than the previous research. The results seem to be even a bit better than in the approach proposed by [9], however, as it was already mentioned, the results cannot be directly compared due to the fact that the evaluation of each method used its own testing dataset and moreover the datasets have different sizes.

Chapter 5

Conclusion

In this text, a novel biometric system for recognition according to the hand geometry was proposed. Together with already well known 2D geometric information, which is used in industrial devices, e.g. HandKey II, etc., a 3D camera is used to obtain also the 3D geometric information and use it in order to improve the performance of the system. The system reuses some of the principles that were already carried out by the previous research in the field of the 3D hand based biometric systems. However, apart from the previous research, this solution tends to use hardware and methods that are applicable in the real world applications.

Therefore, instead of preferirng precision of the acquisition device, its price is of far higher importance. The higher inaccuracy of the acquisition device is compensated by so-called positioning loop that was created to restrict the position of the hand in front of the sensor. Also, in order to use the device properly, the users have to be instructed how to do so.

Speaking about the processing methods, namely feature extraction and matching, this solution is based on existing methods. The 2D measurements are done reusing the approaches proposed in the past. On the other hand, 3D measurements are based on the existing research, however particular changes were done in order to be applicable also for the less precise 3D cameras. Both types of information, i.e. 2D and 3D, are in the end combined using either score normalization methods or metric learning.

For testing, a database of 1033 samples from 100 individuals was collected. In order to evaluate the system behavior and estimate the important parameters, a few subsets of this database were created and particular tests were done. For the training of the LMNN metric learning transformation matrix, again, a small training subset of the whole database was used.

First of all, it was shown which features are valuable and which are not. The less valuable features were removed from the final 2D and 3D feature vectors. Afterwards, evaluation of both score normalization methods and metric learning approach was done on the subset of the whole database first. It has shown that both approaches are usable and provide acceptable results. However, using metric learning approach, the performance of the system can be improved in comparison to the score normalization methods.

Another important thing that was analyzed is the sensitivity of the system to the different transformations of the hand under the sensor. Those transformations are partially avoided by having already mentioned positioning loop, but still, some transformations may occur. The analysis has shown that when the transformations are present in the samples, the system is becoming more unsure about the results, however if the transformations are

low enough, the system still gives acceptable results. When the transformations are bigger, usually on the edge of the correct positioning, the system starts to get confused. One way or the other, these transformations should be avoided as much as possible.

In the end, the evaluation of the system on the whole database was done. The parameters of the system were trained on a small training subset of the whole database. In case of score normalization methods, only fusion of 2D + 3D hand geometric information was evaluated. In this case, the EER is approximately 2.15%, which is generally very good. Using the metric learning approach, the performance was evaluated for three particular cases, 2D only, 3D only and fusion of 2D + 3D information. In case of 2D only, the EER is approximately 2.5%, in case of 3D only, approximately 2.7%. However, fusion of both 2D + 3D geometric information using metric learning gives the EER of approximately 1.42%, which is very good result.

The performance of the system is comparable to the previously done research, as shown in the last subsection of this text. However, as mentioned there, all the approaches were evaluated using different datasets, so the comparison is only illustrative and not really reliable.

This solution gives a very good basis for further development of the 2D + 3D hand geometry based biometric systems for industrial applications. However, as it can be seen, there are still many flaws and questions. Therefore, future work on the project will include further evaluation of the selected features and improvement of their extraction in order to make it more robust. Also, another fusion possibilities will be analyzed. Another task will be to make the system more robust to the effects of different hand transformations as well as improving the positioning loop. Different sensors as alternatives to the one currently used will be tested as well.

All the planned work, experiments and improvements will be done with the goal to make the device a real commercial product that can be put onto the market and create another alternative to the already existing biometric systems.

Bibliography

[1] I. Csiszar and P. C. Shields. Information theory and statistics: A tutorial. *Foundations and Trends in Communications and Information Theory*, 1(4):417–528, 2004. ISSN 1567-2328.

[2] M. Drahansky, F. Orsag, and et al. *Biometrie*. Computer Press a.s., 2011. ISBN 978-80-254-8979-6.

[3] Python Software Foundation. Python. `<https://www.python.org/>`.

[4] M. Gelautz, M. Bleyer, L. He, and N. Brosch. Evaluation and design of energy functions for global stereo matching. `<https://www.ims.tuwien.ac.at/projects/stereo-matching/>`.

[5] Applied Informatics Software Engineering GmbH. Poco c++ libraries. `<http://pocoproject.org/>`.

[6] R. I. Hartley and P. Sturm. Triangulation. *CVIU - Computer Vision and Image Understanding*, 68(2):146–157, 1997. ISSN 1077-3142.

[7] A. K. Jain and N. Duta. Deformable matching of hand shapes for verification. *ICIP - International Conference on Image Processing*, pages 857–861, 1999. ISSN 1522-4880.

[8] A. K. Jain, A. Ross, and S. Pankanti. A prototype hand geometry-based verification system. *Proceedings of AVBPA, Washington DC*, pages 166–171, 1999.

[9] V. Kanhangad, A. Kumar, and D. Zhang. Combining 2d and 3d hand geometry features for biometric verification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 39–44, 2009.

[10] V. Kanhangad, A. Kumar, and D. Zhang. Contactless and pose invariant biometric identification using hand surface. *IEEE Transactions On Image Processing*, 20(5):1415–1423, 2011. ISSN 1057-7149.

[11] R. Klette, K. Schluns, and A. Koschan. *Computer Vision: Three-Dimensional Data from Images*. Springer, 1998. ISBN 9813083719.

[12] B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2013. ISSN 1935-8245.

[13] A. Kumar, D. C. M. Wong, H. C. Shen, , and A. K. Jain. Personal verification using palmprint and hand geometry biometric. *Proceedings of AVBPA, Guildford, U.K*, pages 668–675, 2003.

[14] R. Laganičre. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, Ltd., 2011. ISBN 978-1-849513-24-1.

[15] S. Malassiotis, N. Aifanti, and M. G. Strintzis. Personal authentication using 3-d finger geometry. *IEEE Trans. Info. Forensics and Security*, 1:12–21, 2006. ISSN 1556-6013.

[16] A. A. Puntambekar. *Advanced Data Structures*. Technical Publications, 2007. ISBN 978-8-184-31269-0.

[17] A. A. Ross, K. Nandakumar, and A. K. Jain. *Handbook of Multibiometrics*. Springer, 2006. ISBN 978-0-387-22296-7.

[18] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[19] Inc. The MathWorks. Matlab. `<http://www.mathworks.com/products/matlab/>`.

[20] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10:207–244, 2009.

[21] D. L. Woodard and P. J. Flynn. Finger surface as a biometric identifier. *CVIU*, 100:357–384, 2005. ISSN 1077-3142.

[22] Y. Xu and D. Aliaga. High-resolution modeling of moving and deforming objects using sparse geometric and dense photometric measurements. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1237–1244, 2010.

[23] S. Zhang and S.-T. Yau. High-speed three-dimensional shape measurement system using a modified two-plus-one phase-shifting algorithm. *Opt. Eng.*, 46(11), 2007.

Appendix A

Capture 3D output files

The structure of the PLY file that stores the hand vertices is shown below. Each vertex consists only of three floating point values that are its x-, y- and z-coordinates.

```
ply
format ascii 1.0
element vertex 96367          ; Number of vertices in the model
property float x              ; Vertex elements definition
property float y
property float z
end_header                    ; End of ply header
0.997179 2.72276 6.49693      ; Vertex data
1.0066 2.70994 6.46569
...
```

Model point cloud vertices are also stored in the YAML format, which allows for easy serialization and deserialization of the data. The structure of the YAML file follows.

```
%YAML:1.0
vertices: !!opencv-matrix
   rows: 480
   cols: 640
   dt: "3f"
   data: [ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           ...
           0., 0. ]
```

Elements of the YAML file are easy to understand. It stores an OpenCV matrix in the element called *vertices*. The matrix is described by number of rows, number of columns, data type of each element and then by array of the matrix data itself.

Appendix B

Implementation details

This appendix presents important implementation details that are worth to be mentioned to the reader.

B.1 Used libraries and SDKs

To avoid spending a lot of time reinventing the wheel, well known computer vision and image processing libraries were used wherever possible. In order to have a support for events, memory management, etc. some other libraries were used as well. This section continues with a short description of each library that was used and provides some references to get more information.

OpenCV

OpenCV is an abbreviation for the Open Source Computer Vision that is a powerful library for programming computer vision and image processing applications. It consists of several modules and provides functions for image transformations and segmentation, feature detection, camera calibration and much more.

In this project, OpenCV is generally used for working with matrices and vectors as well as for all the image processing that has to be done.

For more information about OpenCV, please refer to [14].

Poco

The library called Poco is one of the choices when it comes to making a cross-compatible network or Internet applications using the C++ programming language. It is also a good choice when designing applications targeted for an embedded device. It provides classes for dynamic typing, memory management, events and notifications, etc.

This solution takes advantage of Poco in order to ease the memory management by using smart pointers. It also uses the Poco notification system.

More information about Poco can be found at [5].

PCL

PCL stands for Point Cloud Library that is a standalone open source API for 2D or 3D image and point cloud processing. It is distributed under a BSD license. The API is cross-platform and was successfully used on Linux, MacOS, Windows, Android and iOS operating

systems. PCL is composed of many smaller modules that can be compiled independently, which makes it possible to use PCL also on platforms with limited resources.

PCL is used in this project for processing 3D hand point cloud data, in order to compute normals and curvatures, etc.

In order to get more detailed information about PCL, please visit [18].

B.2 Data acquisition

As mentioned in Section 3.1, this project uses the prototype of Intel RealSense camera that is currently being developed by Intel. In order to be able to operate with the camera, the internal Intel RealSense library is used. This library allows to obtain both color and depth streams from the 3D camera. It also provides support for projection operations to get a 3D point cloud out of the depth map image.

Intel RealSense 3D camera streaming

The camera depth streaming provides the IR intensity image and the depth map image. It is run with the following parameters:

- resolution 640×480;
- 60 frames per second;
- IR intensity format UINT8 (range 0 - 255);
- depth map format UINT16 (range 0 - 65535).

Using the projection functions provided by the Intel RealSense (IVCAM) library, a point cloud composed of the floating point 3D vertex coordinates is obtained from the depth map.

With every frame in the depth stream, a notification that contains the *SensorOutput* structure data is sent. The *SensorOutput* structure is defined as follows:

```
struct SensorOutput {
    cv::Mat depth;          // UINT16 2D depth map image
    cv::Mat intensity;      // UINT8 2D IR intensity image
    cv::Mat vertices;       // FLOAT 3D vertices
};
```

Camera stream processing

As noted in the previous paragraph, with every new frame in the camera stream, a notification that contains the sensor output data is sent. There is an input processor that has an observer registered for that notification. With every notification it receives, it analyzes the input data carried by the notification and processes it accordingly. The communication between the IVCAM stream provider and the input processor is illustrated in Figure B.1.

The input processor implements so-called "positioning loop" that takes the input images and processes them in order to analyze the hand position in the image. According to the current hand position it either indicates that the hand is positioned correctly or it leads the user how to correct the hand position.

Before the hand position can be analyzed, some points of interest have to be detected first. For the purpose of position check, first the hand contour, the approximate position of

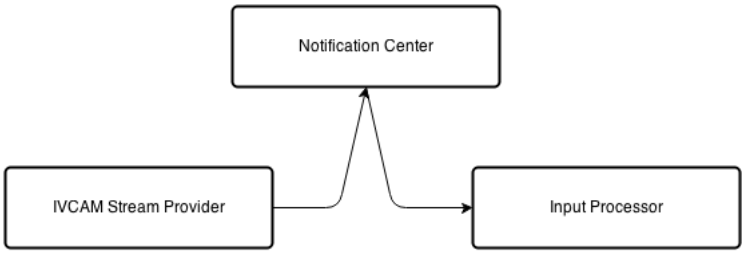


Figure B.1: Connection between IVCAM stream provider and input processor via notification center.

the hand centroid and the fingertips are found. Methods used to get those information are the same as the ones used during the feature extraction process. For more details about these methods, please see Section B.4.

Having the required points of interest, the hand position for the current frame is analyzed. The diagram in Figure B.2 illustrates how the analysis is done.

B.3 Hand features representation

A few structures were designed in the application in order to describe the human hand surface using the points of interest described earlier in Section 3.3.

These structures can be basically divided into two groups. The first group consists of structures that encapsulates the positions of the interest points. The second group stores the actual features, which means the distances computed during the analysis of the hand surface. Both groups are described in more detail in the following two subsections.

Hand surface interest points

The hand surface is described by a set of structures that contain x- and y- coordinates of the interest points as well as indices of those points in the hand contour array.

The three basic structures are FingerTip structure, FingerValleys structure and WristLine structure and they are described next.

The first of the structures, the **FingerTip** structure is used to describe the fingertip of a particular finger. As you can see in the UML diagram in Figure B.3, it consists of two components. The pt component is a 2D point that represents the fingertip 2D coordinates and the *index* is the index of the fingertip point in the hand contour array.

The second of the three basic structures is the **FingerValleys** structure. It describes the finger valleys corresponding to a particular finger and its UML diagram is shown in Figure B.4. In the diagram, the pts is an array of two elements that contains 2D coordinates of the right and the left finger valley point of a particular finger, the *indices* is an array of integers that contains indices of the finger valley points in the hand contour array and the *center* point is the middle point of the line that is connecting the two finger valley points.

The **WristLine** structure is the last one of the three basic structures and it is shown as an UML diagram in Figure B.5. The wrist line is described by two points on the hand wrist that represents the starting and the ending point of the line. The 2D coordinates of those points are represented by $ptLeft$ and $ptRight$ in the WristLine structure.

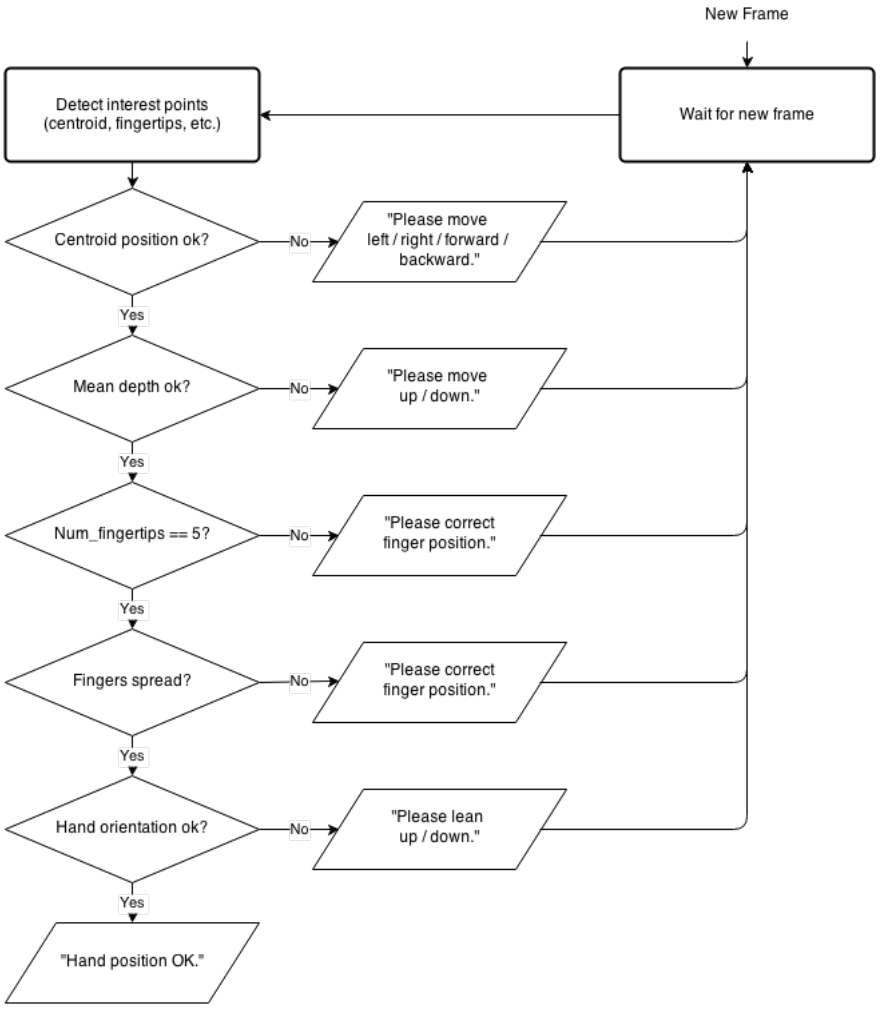


Figure B.2: The positioning loop is used to lead the user towards the proper hand position.

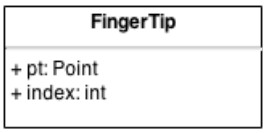


Figure B.3: UML diagram of the FingerTip structure.

To describe all the fingers on the hand surface, two more structures are created using the previously defined structures. Those are namely the Finger structure and the Fingers structure. Definitions of those structures are described next.

The **Finger** structure is designed to group all the properties of one finger. The structure is visualized as UML diagram in Figure B.6. Therefore, the *tip* component is describing the fingertip, the *valleys* component the finger valley points and the *axis* is the direction vector of the finger axis.

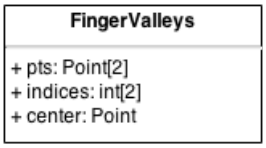


Figure B.4: UML diagram of the FingerValleys structure.

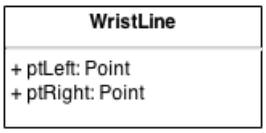


Figure B.5: UML diagram of the WristLine structure.

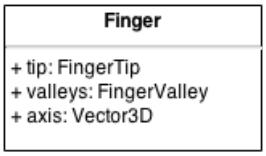


Figure B.6: UML diagram of the Finger structure.

The **Fingers** structure is then grouping the Finger structures for all the five fingers of a hand and its UML diagram is shown in Figure B.7.

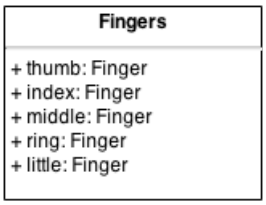


Figure B.7: UML diagram of the Fingers structure.

The hand surface is in the end described by taking all the previously defined structures and creating one final structure that contains all the feature points. This structure is called **Hand** and it is defined by the UML diagram in Figure B.8. In the UML diagram, the *centroid* is the centroid point of the hand surface.

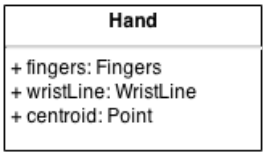


Figure B.8: UML diagram of the Hand structure.

During each input processing, the structures introduced above are taken and filled with information. The output of feature detection is therefore the **Hand** structure, which describes the whole hand surface using the important features.

However, for the purpose of matching, such a description is not sufficient and therefore, another set of structures was defined. Those are described in the following subsection.

Hand surface distances

During the feature extraction process, the structures defined in the previous subsection are taken and particular distances are computed on the hand surface. In order to store those distances in some readable objects, a few structures for representation of the distances were defined.

One structure was defined for the representation of the finger lengths and therefore it is called **FingerLengths** structure. Its definition is represented by the UML in Figure B.9.

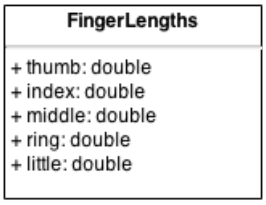


Figure B.9: UML diagram of the FingerLengths structure.

Another structure called **FingerWidths** was defined in order to represent the finger widths measured over the hand surface at a particular position on the fingers. It is described as UML in Figure B.10. The $valleyDist$ is the distance between finger valley points, w is an array of eight measured width values, the $w3D$ is the array of the cross sectional segments, i.e. the widths, measured on the 3D model. The last component, the $axisSurfaceDist$ 2D array is an array of the five distances from the surface to the cross sectional segment finger axis measured on the last six cross sectional segments of the finger.

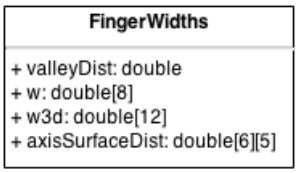


Figure B.10: UML diagram of the FingerWidths structure.

The next one is the **Widths** structure, which was defined in order to encapsulate the FingerWidths structures for all five fingers. It is represented by the UML in Figure B.11.

The last structure is called **Wrist** and it is used to store the distances related to the wrist of the hand. It is described by the UML diagram shown in Figure B.12, where the $wristWidth$ is the width of the wrist of the hand and $wristValley$ is array of few measured distances between particular wrist points and finger valley points.

All the four structures introduced above are encapsulated in one more structure called **Hand** that stores all the distances computed on the hand surface. This structure also

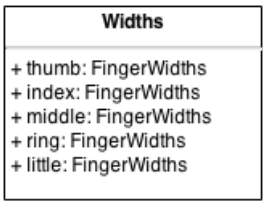


Figure B.11: UML diagram of the Widths structure.

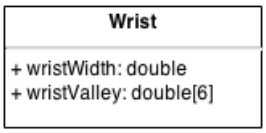


Figure B.12: UML diagram of the Wrist structure.

provides methods to create one dimensional feature vectors that consist of all the computed distances. It is represented as UML in Figure B.13. Description of particular elements of the feature vector can be found in Section 3.5.

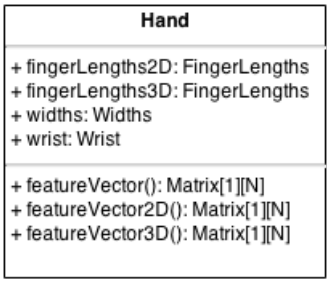


Figure B.13: UML diagram of the Hand structure for the distances.

B.4 Details on feature extraction

Detection and extraction of the hand features consist of a few steps that have to be done in a particular order.

First of all, the hand contour is extracted. Based on the hand contour information, the hand centroid point is detected. Using the hand contour and the hand centroid point, the fingertip points are found. Next, based on the fingertip position knowledge, the finger valley points can be computed. In the end, the wrist line position is estimated based on the already obtained information.

All the methods created for the purpose of feature detection take advantage of the OpenCV and PCL libraries mentioned earlier in Section B.1.

There are few parameters that are set manually and influence the whole feature extraction process. The UML representing **Setup** structure that encapsulates those parameters can be found in the Figure B.14 below. The $depthThresh$ is the depth value threshold

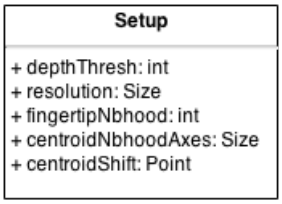


Figure B.14: UML diagram of the Setup structure that stores the parameters of the system.

for the foreground - background separation, *resolution* is the resolution of the 2D input images. The $fingertipNbhood$ stands for the diameter of the fingertip neighborhood area in pixels. The parameter $centroidNbhoodAxes$ holds the lengths of the axes of the elliptical hand centroid neighborhood in the units of pixels. The point $centroidShift$ represents the shift of the centroid point along the x- and y-axes.

Those parameters are set by the user before the processing starts. It is very important to set them properly, otherwise the feature detection process may fail.

Fingertip detection

As an input, the fingertip detection algorithm receives the hand contour and the hand centroid point. At the beginning, the hand contour is approximated by convex hull using OpenCV function cv::convexHull(...). Next, fingertip candidates are filtered using the centroid point and they get assigned an angle that approximately represents their curvature. Those points are segmented into multiple groups using OpenCV function cv::partition(...). In each cluster, the point with the lowest x-coordinate is taken as the final candidate representing the cluster. If there are more than five clusters after the segmentation, the representatives of the five clusters with the lowest x-coordinates are taken as the fingertips. In the end, a check against point with a better curvature is done in close proximity of each fingertip. If there is such a point, it is exchanged and set to be the fingertip instead. The whole process is represented by the Algorithm 1.

When all the five fingertips are found, their indices in the hand contour array are extracted and they are saved together with the fingertip coordinates.

Finger valleys detection

In addition to the hand contour and the hand centroid point, in case of the finger valleys detection algorithm, the fingertip points are on the input as well. For each two fingertip points, the hand contour area between them is traversed and point with the highest x-coordinate is taken as a finger valley candidate, if it lies within a defined distance from the hand centroid point. This way, four main finger valley points are detected. As described in Section 3.4, based on the four main finger valley points, another three finger valley points are obtained to get the full description of all the fingers. The finger valley detection process is described by the Algorithm 2.

As well as in the case of fingertips, after all the finger valleys are found, their indices in the hand contour array are computed too and they are both stored.

Algorithm 1 Fingertip detection

```
1: FindConvexHull()
2: for all points in convexHull do
3:    if point.x ¡ handCentroid.x then
4:       CurvatureEstimate(point)
5:       if curvature big enough then
6:          fingertipCandidates.add(point)
7:       end if
8:    end if
9: end for
10: PartitionIntoClusters(fingertipCandidates)
11: for all clusters do
12:    GetPointWithLowestXCoordinate(clusterPoints)
13:    fingertips.add(pointWithLowestXCoord)
14: end for
15: for all fingertips do
16:    CheckFingertipNeighborhoodCurvatures(fingertipPoint)
17: end for
```

Algorithm 2 Finger valleys detection

```
1: for all fingertips do
2:    GetContourPointsSegment(fingertip[i], fingertip[i + 1])
3:    for all points on contour segment do
4:       if point.x ¿ candidate.x then
5:          candidate = point
6:       end if
7:    end for
8:    if IsCandidateInRange(candidate, handCentroid) then
9:       valleyPoints.add(candidate)
10:   end if
11: end for
12: if valleyPoints.size() == 4 then
13:    ComputeAdditionalFingerValleys()
14: end if
```

Curvature estimation

A really simple approach was used for the curvature estimation for a given point on the hand contour surface in order to verify that the point is not lying on a flat part of the contour.

There is no need for precision in this case. Therefore, for each point with the index i that the curvature should be computed for, its neighbor points from the left and from the right with indices $i - 30$ and $i + 30$ are taken and two vectors are computed. The first vector is defined by taking points with the indices i and $i - 30$, the second one is defined by the points with the indices i and $i + 30$. In the end, the angle between the two vectors is computed and taken as a value representing the curvature at the point with index i.

3D length computation

Various lengths are computed as the 3D features on the 3D hand model surface. The computation of a 3D length completely differs from the computation of a length on the 2D hand contour.

For every point on the hand in the 2D image, its coordinates in the 3D hand model are known. This way, the 3D lengths are computed for the same segments as the 2D lengths on the hand contour. In order to compute the 3D length, a line is defined in the 2D image. For each point on the defined line, its 3D coordinates in the 3D hand model are taken and distances between the neighbor ones are computed. Those distances are in the end summed up in order to get the resulting 3D length. This procedure is shortly described by Algorithm 3.

Algorithm 3 3D length computation

```
1: DefineLineIn2DImage()
2: Get3DCoordinates(linePoints)
3: finalDistance = 0
4: for all coordinates in linePoints3DCoordinates do
5:     dist = ComputeDistance(coordinates[i], coordinates[i + 1])
6:     finalDistance = finalDistance + dist
7: end for
```

B.5 Matching feature vectors

The implementation of the feature matching is very much simplified by taking advantage of OpenCV support for working with vectors and matrices.

As mentioned in Section B.4, after the feature detection, a structure describing the hand surface is available. For the purpose of comparison, this structure is converted into the feature vector. The feature vector is always represented by the OpenCV matrix (cv::Mat), which has one row and the number of columns corresponds to the number of components of the feature vector.

Section 3.5 describes different methods for the feature vector comparison. The implementation of those methods is shortly described below.

Mahalanobis distance

One of the possible solutions is that the comparison of the feature vectors is done using the Mahalanobis distance as said in Section 3.5. The equation for the Mahalanobis distance computation is easily expressed using the OpenCV matrix arithmetic operations as shown below.

$$(\mathrm{fv}_1 - \mathrm{fv}_2) \cdot \mathrm{scale} \cdot (\mathrm{fv}_1 - \mathrm{fv}_2)^T \tag{B.1}$$

In the computation B.1 above, fv_1 and fv_2 are two matrices with the size of $1 \times N$ and the *scale* is an inverted diagonal matrix of size $N \times N$ carrying the feature vector components weights.

The vector of weights is loaded from a CSV file using the OpenCV ML (Machine Learning) module that provides easy support for loading matrices stored in the CSV file format. This vector of weights is then converted into a square diagonal matrix using function cv::diag. As mentioned in the previous article, the *scale* matrix is an inverted diagonal matrix. When the matrix is diagonal, it can be easily inverted by just inverting every single element on the diagonal, which is done in this case.

Basically the same applies when it comes to 2D and 3D matching score fusion. It is again very easily expressed taking advantage of the OpenCV support for matrix and vector arithmetic.

Metric learning based transformation

The second solution uses a transformation matrix that was trained using the LMNN metric learning approach as explained in Section 2.3. The learning process itself was described in Section 4.1.5. It is assumed that a CSV file, which stores the transformation matrix, is available. As well as in the previous subsection, the OpenCV ML module was used in order to load the transformation matrix from the CSV file.

Apart from the first method, in this case the feature vectors are expected to have the size $N \times 1$. The application by default works with the feature vectors of size $1 \times N$ and therefore, before the distance computation itself is performed, the vectors are transposed to get the vector with the required dimensions.

In the end, the distance is computed using the equation shown earlier in Section 3.5.1, which can be easily implemented with the help of the OpenCV library again. The computation is shown below.

$$(\mathrm{fv}_1 - \mathrm{fv}_2)^T \cdot \mathrm{transf}^T \cdot \mathrm{transf} \cdot (\mathrm{fv}_1 - \mathrm{fv}_2) \tag{B.2}$$

In the Equation B.2 fv_1 and fv_2 are the transposed input feature vectors and *transf* is the loaded transformation matrix.

Appendix C

Used testing tools

When it comes to experiments, it usually means evaluating the same equations many times with different parameters in order to obtain the best results. For that purpose, it is ideal to create some scripts that does the evaluation automatically without repeating it manually every time the parameters are changed.

In case of this project, Python[3] scripting language and Matlab[19] were used. Their purpose is shortly described in the following two subsections.

Python

During the experiments, a bigger database of users is used for the evaluation of the system. However, the database of users is provided as a folder structure where each folder contains images from one person. For the evaluation purposes, it is required to merge all the data into one folder and label them. This is where Python comes in handy.

The Python scripting language was used mainly for the following two reasons. It provides good structures for easy data labeling and moreover it has a good module for platform independent management of files and directories. Using this module, folder structure that contains the input data is traversed and merged into one folder. During the folder traversal, data labelling is performed too.

Labeled data are in the end described by four text files. The following list shortly describes the text files:

- **all.txt** - contains all samples with their labels. One row corresponds to one sample. Each row has the format „imageNum imageLabel“;
- **labels.txt** - contains the image labels and corresponding user IDs. One row corresponds to one image label. The row format is „imageLabel userID“;
- **train.txt** - has the same format as all.txt, but includes only samples that were used for the training;
- **test.txt** - also has the same format as all.txt, but includes only samples that were used for the testing.

Matlab

Matlab is the right tool to be used when it comes to statistical data analysis, data visualization, etc.

Apart from Python, which was used in order to prepare the input data, Matlab was used for evaluation of the output data, which are either the extracted features (i.e. distances) or the matching scores.

It was also used for the metric learning process. For this task, specifically Matlab implementation of the LMNN by Kilian Q. Weinberger was used. For more information about the implementation, please refer to [20].

The input data for created Matlab scripts were always stored as a CSV file. List of the input files for the Matlab evaluation is shown below.

- **featureVectors2D.csv** - extracted feature vectors for the 2D features. One row corresponds to one feature vector. Each row has the following format „label fvComponent1 ... fvComponentN“;
- **featureVectors3D.csv** -the same as the featureVectors2D.csv, but in this case for the 3D features;
- **scores2D.csv** - obtained scores from 2D feature vector comparison for 2D matching score normalization;
- **scores3D.csv** -obtained scores from 3D feature vector comparison for 3D matching score normalization;
- **genuine.csv** - scores for genuine samples for threshold estimation;
- **impostor.csv** - scores for impostor samples for threshold estimation.