VIEWPOINT INVARIANT GESTURE RECOGNITION AND 3D HAND POSE ESTIMATION USING RGB-D

by

PAUL DOLIOTIS

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington, in the context of the joint PhD programme with the National Center for Scientific Research ”Demokritos”, in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2013

ACKNOWLEDGEMENTS

First of all I would like to express my gratitude to my supervisor Dr. Vassilis Athitsos for his invaluable support and guidance during the course of my PhD studies and especially for tolerating my idiosyncrasies and offering me the flexibility to follow my own research agenda. His scientific ethos has been an inspiration from the very beginning and was pivotal in forging my own scientific identity. I would also like to thank my committee members, Dr. Stavros Perantonis, Dr. Gerasimos Potamianos, Dr. Farhad Kamangar and Dr. Gian -Luca Mariottini, for their interest in my research, for participating in my defense and for providing helpful comments and insights that improved the overall quality of this dissertation.

I had the wonderful opportunity to be the first scholar of the joint PhD programme between the University of Texas at Arlington and the National Center for Scientific Research "Demokritos". During the last five years I had the privilege to collaborate with researchers from UTA (VLM lab) and Demokritos (CIL lab), taking the best of both worlds. I wish to thank all of my colleagues for creating such a fun and stimulating working/research environment. Dr. Filia Makedon (from UTA) and Dr. Vangelis Karkaletsis (from Demokritos) played an instrumental role in establishing this joint PhD programme and I thank both of them. I would also like to thank my co-advisors, Dr. Stavros Perantonis and Dr. Dimitrios Kosmopoulos, for being frequent research collaborators and for their support during the time I spent as a research assistant at the CIL lab in Demokritos.

Finally, last but not least, I am grateful to my family for their unwavering moral support for more than a decade of academic studies.

November 15, 2013

ABSTRACT

VIEWPOINT INVARIANT GESTURE RECOGNITION AND 3D HAND POSE ESTIMATION USING RGB-D

PAUL DOLIOTIS, Ph.D.

The University of Texas at Arlington, 2013

Supervising Professor: Vassilis Athitsos

The broad application domain of the work presented in this thesis is pattern classification with a focus on gesture recognition and 3D hand pose estimation.

One of the main contributions of the proposed thesis is a novel method for 3D hand pose estimation using RGB-D. Hand pose estimation is formulated as a database retrieval problem. The proposed method investigates and introduces new similarity measures for similarity search in a database of RGB-D hand images. At the same time, towards making 3D hand pose estimation methods more automatic, a novel hand segmentation method is introduced which also relies on depth data. Experimental results demonstrate that the use of depth data increases the discrimination power of the proposed method.

On the topic of gesture recognition, a novel method is proposed that combines a well known similarity measure, namely the Dynamic Time Warping (DTW), with a new hand tracking method which is based on depth frames captured by Microsoft's Kinect™ RGB-Depth sensor. When DTW is combined with the near perfect hand tracker gesture recognition accuracy remains high even in very challenging datasets, as demonstrated by experimental results. Another main contribution of the current thesis is an extension of the

proposed gesture recognition system in order to handle cases where the user is not standing fronto-parallel with respect to the camera. Our method can recognize gestures captured under various camera viewpoints.

At the same time our depth hand tracker is evaluated against one popular open source user skeleton tracker by examining its performance on random signs from a dataset of American Sign Language (ASL) signs. This evaluation can serve as a benchmark for the assessment of more advanced detection and tracking methods that utilize RGB-D data. The proposed structured motion dataset of (ASL) signs has been captured in both RGB and depth format using a Microsoft Kinect^{TM} sensor and it will enable researchers to explore body part (i.e., hands) detection and tracking methods, as well as gesture recognition algorithms.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In human-computer interaction applications, gesture recognition has the potential to provide a natural way of communication between humans and machines. The technology is becoming mature enough to be widely available to the public and real-world computer vision applications start to emerge. However human-computer interaction interfaces need to be as intuitive and natural as possible. The user should ideally interact with machines without the need of cumbersome devices (such as colored markers or gloves) or apparatus like remote controls, mouse and keyboards. Hand gestures can provide an alternative and easy means of communication with machines and could revolutionize the way we use technology in our daily activities. Successful applications of hand gesture systems can be found in various research and industry areas such as: game controlling, human-robot interaction, virtual environments, smart homes and sign language recognition, to name a few.

The broad application domain of the work presented in the following chapters is pattern classification with a focus on viewpoint invariant gesture recognition and 3D hand pose estimation using RGB-D which are formulated as database retrieval problems. The user provides to the gesture recognition system examples, and asks the system to retrieve database items that are the most similar to those examples. The system achieves classification of that example based on the class labels of the most similar database patterns.

The main contributions of this dissertation can be summarized as follows:

1. A viewpoint invariant hand pose estimation method using RGB-D (Chapter 2)

2. A hand tracking method based on depth data which is evaluated against one popular user skeleton tracker by examining its performance on random signs from a dataset of American Sign Language (ASL) signs. (Chapter 3)
3. An end-to-end gesture recognition system (see Chapter 4) that uses RGB-D and combines a well known similarity measure, namely the Dynamic Time Warping (DTW), with a new hand tracking method which is based on depth frames.
4. A viewpoint invariant gesture recognition method that can handle cases where the user is not standing fronto-parallel with respect to the camera (Chapter 5).

More specifically, the first main contribution of the thesis is a viewpoint invariant hand pose estimation method using RGB-D (see Chapter 2). It proposes an exemplar-based method that relies on similarity measures employing depth information. Our system, given an input image of a person signing a gesture in a cluttered scene, locates the gesturing arm, automatically detects and segments the hand and finally creates a ranked list of possible shape classes, 3D pose orientation and full hand configuration parameters. The clutter-tolerant hand segmentation algorithm is based on depth data from a single image captured with a commercially available depth sensor, namely the Kinect^{TM}. Shape and 3D pose estimation is formulated as an image database retrieval method where given a segmented hand the best matches are extracted from a large database of synthetically generated hand images. Contrary to previous approaches this clutter-tolerant method is all-together: user-independent, automatically detects and segments the hand from a single image (no multi-view or motion cues employed) and provides estimation not only for the 3D pose orientation but also for the full hand articulation parameters. The performance of this approach is quantitatively and qualitatively evaluated on a dataset of real and synthetic images of American Sign Language (ASL) handshapes.

Another main contribution, is an exemplar-based system for gesture recognition which is presented at Chapter 4. A novel method is proposed that combines a well known

similarity measure, namely the Dynamic Time Warping (DTW), with a new hand tracking method which is based on depth frames captured by Microsoft's Kinect™ RGB-Depth sensor. First we evaluate our depth hand tracker (see Chapter 3) against one popular user skeleton tracker by examining its performance on random signs from a dataset of American Sign Language (ASL) signs. Our structured motion dataset of (ASL) signs has been captured in both RGB and depth format using a Microsoft Kinect™ sensor and it will enable researchers to explore body part (i.e., hands) detection and tracking methods, as well as gesture recognition algorithms. The proposed gesture recognition system relies on the accurate depth hand tracker and is one of the earliest ones that employed such depth information from the Kinect™ sensor. The underlying gesture recognition method is translation and scale invariant which is a desirable property for many HCI systems. Performance has been tested on a digits recognition dataset which has been captured in a rather challenging environment with clutter in the background as well as various moving distractors that could make typical gesture recognition systems fail . All experimental datasets include hand signed digits gestures but the framework can be generalized to recognize a wider range of gestures.

At Chapter 5 we extend our recognition system in order to handle cases where the user is not standing fronto-parallel with respect to the camera. Our viewpoint invariant gesture recognition method can recognize gestures captured under various camera viewpoints, in the range of $[-75^\circ \cdots + 75^\circ]$. A few interesting properties of our system are the following:

1. It is trained from videos captured under one specific camera viewpoint but it can be tested with gestures captured under arbitrary camera viewpoints. In our experiments we opt to train our system with a camera viewpoint where the user is standing fronto-parallel to the image plane. For testing the videos are captured under the following set of viewpoints $\{\pm 45^\circ, \pm 75^\circ\}$.

2. It is all-together translation, scale and viewpoint invariant. To the best of our knowledge few gesture recognition methods satisfy all these three properties at the same time.
3. It employs an affordable, commercially available sensor (i.e., Microsoft Kinect^{TM}) as opposed to an expensive laboratory sensor or a cumbersome calibrated multi-camera set-up.

In the upcoming chapters we will further elaborate on the proposed contributions and we will provide experimental results that demonstrate the usefulness and effectiveness for all novel methods presented throughout this thesis.

CHAPTER 2

3D HAND POSE ESTIMATION USING RGB-D

2.1 Introduction

This chapter will investigate and propose novel similarity methods that are integrated in the general framework of 3D hand pose estimation. Hand pose estimation belongs to the broader application domain of gesture recognition and has become an essential component for many natural user interface (NUI) systems. It provides humans the ability to interact with machines naturally without the use of any cumbersome mechanical devices. Hand gestures are more commonly used and can be found in a wide range of applications such as: sign language recognition, robot learning by demonstration and gaming environments, just to name a few. Recognizing hand gestures is a very challenging task and requires solving several sub-problems like automatic hand detection and segmentation, 3D hand pose estimation, hand shape classification and in some cases estimation of the full hand configuration parameters.

In this work we specifically address the problem of 3D hand pose and shape estimation. Towards developing an effective solution several challenges may arise and some of the main ones are listed bellow:

- High dimensionality of the problem
- Noisy hand segmentation due to cluttered backgrounds
- Increased pose variability and self-occlusions that frequently occur when a hand is in motion

Hand pose estimation is formulated here as an image database retrieval problem. The closest matches for an input hand image are retrieved from a large database of synthetic hand

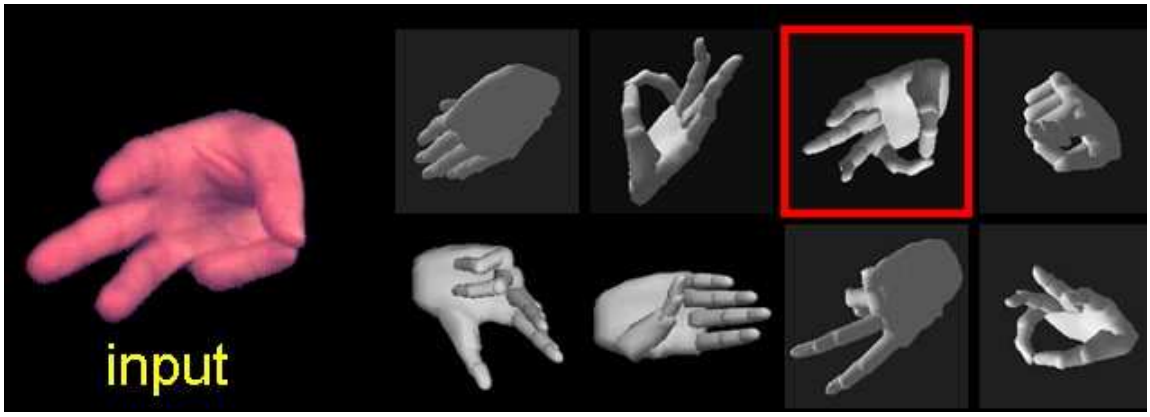


Figure 2.1. System input and output. Given the input image, the system goes through the database of synthetic images in order to identify the ones that are the most similar to the input image. Eight examples of database images are shown here, and the most similar one is enclosed in a red square. The database currently used contains more than 100,000 images.

images. The ground truth labels of the retrieved matches are used as hand pose estimates from the input (Figure 2.1). The approach described in this chapter is motivated by the work presented in [4]. However, one limitation of that work was that it required manual segmentation in order to define a bounding box for the gesturing hand. We propose an automatic hand segmentation method that relies on depth data acquired from the Microsoft Kinect^{TM} device [5]. Another contribution is that we achieve improved performance under clutter by using a similarity measure which is also based on the depth data. A main assumption we make is that the gesturing arm is the closest object to the camera and so it can easily be segmented from the rest of the body and other objects based on depth. To measure the effectiveness of this new method we have collected a dataset of American Sign Language (ASL) handshapes.

2.2 Related Work

Some successful early works require specialized hardware or the use of cumbersome mechanical devices. In [6] Schneider and Stevens use a motion capture system while in [7]

Wang and Popović employ visual markers with a color glove. Unfortunately such methods impede the user's natural interaction in the scene and they require a costly and complex experimental setup.

Nowadays research is more focused on purely vision-based methods that are non-invasive and are more suitable for Natural User Interface (NUI) systems. The most recent review on vision-based hand pose estimation methods has been published by Erol *et al.* [8]. They define a taxonomy where initially these approaches are divided in two main categories: "partial pose estimation" and "full DOF pose estimation". "Partial pose estimation" methods can be viewed as extensions of appearance-based systems. They usually take as input image features and map them a small discrete set of hand model kinematic parameters. A main disadvantage is that they require a large amount of training data and hence are not scalable. Appearance-based methods for hand pose recognition, like [9, 10, 11, 12], can tolerate clutter, but they are limited to estimating 2D hand pose from a limited number of viewpoints. Our method can handle arbitrary viewpoints.

"Full DOF pose estimation" approaches are not limited to a small, discrete set of hand model configurations. They target all the kinematic parameters (i.e., joint angles, hand position or orientation) of the skeleton of the hand, leading to a full reconstruction of hand motion. These approaches can be further divided into two other categories: (1) "Model-based tracking" and (2) "Single frame pose estimation".

"Model-based methods" [13, 14, 15, 16] typically match visual observations to instances of a predefined hand model. Formally this is expressed as an optimization problem where an objective function is required in order to measure similarity between actual visual observations and model hypotheses. The main drawback is increased computational complexity due to the high dimensionality of the model's parameter space. On the other hand they require less training and are easily scalable.

"Single frame pose estimation methods" try to solve the hand pose estimation problem without relying on temporal information. The lack of temporal information increases the difficulty of the problem. However successful approaches can tackle the negative effect of motion blur and can also be employed to initialize tracking-based systems. Athitsos *et al.* [4] have proposed such a single pose estimation method by creating a large database of synthetic hand poses using an articulated model and retrieve the best match from this database. However they require manual segmentation of the test data.

Most recently, due to the advent of commercially available depth sensors, there is an increased interest in methods relying on depth data [17, 18, 13, 19]. Keskin *et al.* [17] train Random Decision Forests (RDF) on depth images and then use them to perform per pixel classification and assign each pixel a hand part. Then, they apply the mean shift algorithm to estimate the centers of hand parts to form a hand skeleton. However they don't explicitly address the automatic hand segmentation problem.

Another highly cited method that relies on RGB-D data has been proposed by Oikonomidis et al. [13]. This is a model-based method that treats 3D hand pose recovery as a minimization problem. The objective function to be minimized is formulated as the difference between a 3D parametric model and the actual instances of captured hand images. The objective function employs both RGB and depth information provided by a Kinect™ sensor. More specifically, the 3D hand model is defined as a set of assembled geometric primitives and is expressed as a vector of 27 parameters. Hand pose estimation and tracking is achieved by computing the optimal values for those 27 parameters that minimize the difference between hand hypotheses and the actual observations. To quantitatively measure that difference a 3D rendering software is employed in order to produce RGB and depthmap instances for given model parameters. The minimization is formulated with a variant of Particle Swarm Optimization. Near real time performance is achieved by exploiting the GPU's parallel processing architecture. This technique requires temporal continuity as

opposed to our proposed hand pose estimation method that relies on information from a single frame. Another main difference is the initialization of the system which is not a requirement in our case.

According to the aforementioned taxonomy, this chapter describes an "appearance-based method" aiming at 3D orientation estimation using features from a single frame. This work builds on top of the work described in [4] and [20], where hand pose is estimated from a single cluttered image. The key advantages of the method described here over [4] are that we integrate an automatic hand segmentation method and we use a similarity measure that is based on depth data. A recent published version of this work can be found in [20].

2.3 Hand Segmentation

As a first step we need to perform a rough segmentation by thresholding the depth data in order to obtain the gesturing arm. Given the assumption that the hand is the closest object to the camera we can automatically find the lower depth threshold. As an upper threshold we take an initial rough estimation, since at this point we are only interested at segmenting the arm. A more precise thresholding is needed however if we need to further segment the hand. To find the palm cutoff point we need to perform the following steps:

1. Compute the axis of elongation for the gesturing arm.
2. Create a sequence of widths.
3. Perform a gradient descent on the sequence of widths in order to identify the local (or global) minimum, at which the palm cutoff point is located.

2.3.1 Finding the Axis of Elongation

The result of the initial rough segmentation is a blob representing the gesturing arm (see Figure 2.2).

Figure 2.2. At the top image we can see the original depth image. At the bottom image we can see the segmented arm after performing depth segmentation using a rough estimation for an initial threshold. Original size for both images is 640×480.

The boundary of that blob is essentially a set $\mathcal{S}$ of m points in 2 dimensional space:

Noisy smaller groups of pixels are usually part of $\mathcal{S}$. To remove them, we morphologically

open the binary image by eliminating all connected components (objects) that have fewer than 20 pixels, considering an 8-connected neighborhood. The remaining boundary pixels will belong to a new set $\mathcal{S}' = \{x'_1, x'_2, \ldots, x'_k\} \in \mathbb{R}^2$. In order to define the elongation axis of the gesturing arm we will compute the *Minimum Enclosing Ellipsoid (MEE)* for the boundary pixels $x'_i \in \mathcal{S}'$. The major axis of the *MEE* coincides with the arm's axis of elongation (Figure 2.3).

An ellipsoid in center form can be given by the following equation:

$$\mathcal{E} = \left\{ x' \in \mathbb{R}^2 | (x' - c)^T E (x' - c) \leq 1 \right\} \tag{2.1}$$

where $c \in \mathbb{R}^2$ is the center of the ellipse $\mathcal{E}$ and E is a 2×2 positive definite symmetric matrix, $E \in \mathbb{S}^2_{++}$.

So finding the Minimum Enclosing Ellipsoid can be formulated as an optimization problem as follows:

$$\begin{aligned} \underset{E,c}{\text{minimize}} \quad & det(E^{-1}) \\ \text{subject to} \quad & (x'_i - c)^T E (x'_i - c) \leq 1, \ i = 1, \ldots, k \\ & E \succ 0 \end{aligned} \tag{2.2}$$

An implementation of a solver based on the Khachiyan Algorithm [21] can be found at the web [22]. The major axis for the arm boundary pixels will coincide with the major axis of the Minimum Enclosing Ellipsoid.

2.3.2 Creating the Sequence of Widths

After the major (or elongation) axis is obtained we can easily create a sequence of widths. In the discrete domain, the elongation axis is comprised of a set of pixels $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$. For each p_i we compute the maximum distance of arm pixels belonging to the line that goes through p_i and it's direction is perpendicular to the direction of the

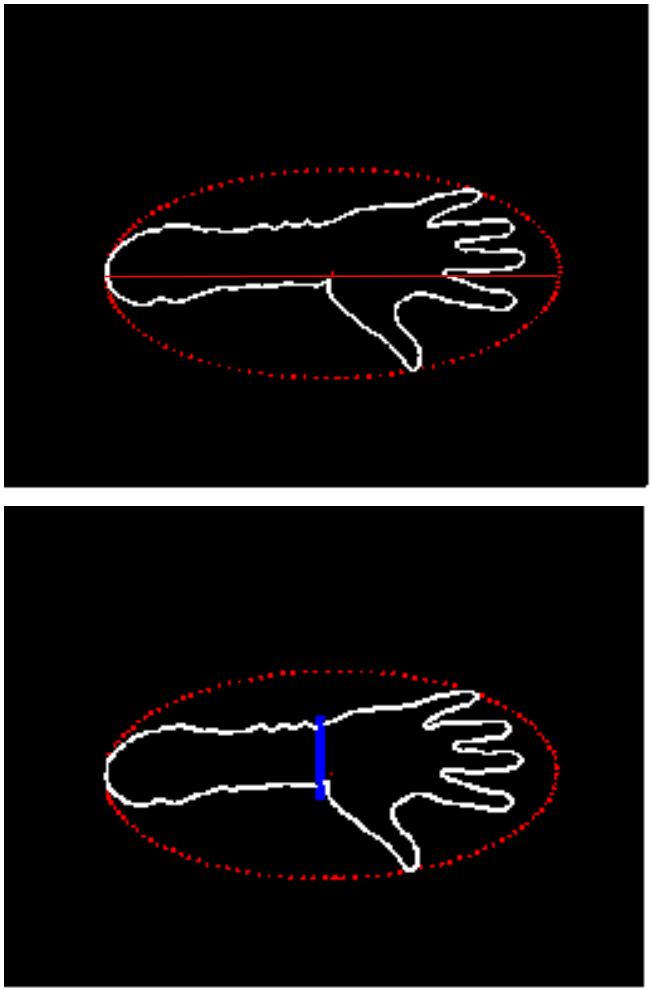


Figure 2.3. On the top is an example of a *Minimum Enclosing Ellipsoid (MEE)* along with the major axis. To the bottom we visually demonstrate the desired palm cutoff location (blue line).

elongation axis. The main idea is that at the palm cutoff point the sequence will reach a global or local minimum. Since the contour of the segmented arm is rugged our method could be prone to other local minima. To alleviate this effect we apply a smoothing on our 2D contour. In Figure 2.5 one can see the effect of smoothing on a 2D contour of a hand.

Smoothing 2D contours is achieved by using *Local Regression Lines*. Because of the linear nature of fitting it might be possible to loose important information in special cases like corners (or fingertips). To tackle this issue we opt to fit locally the line by employing

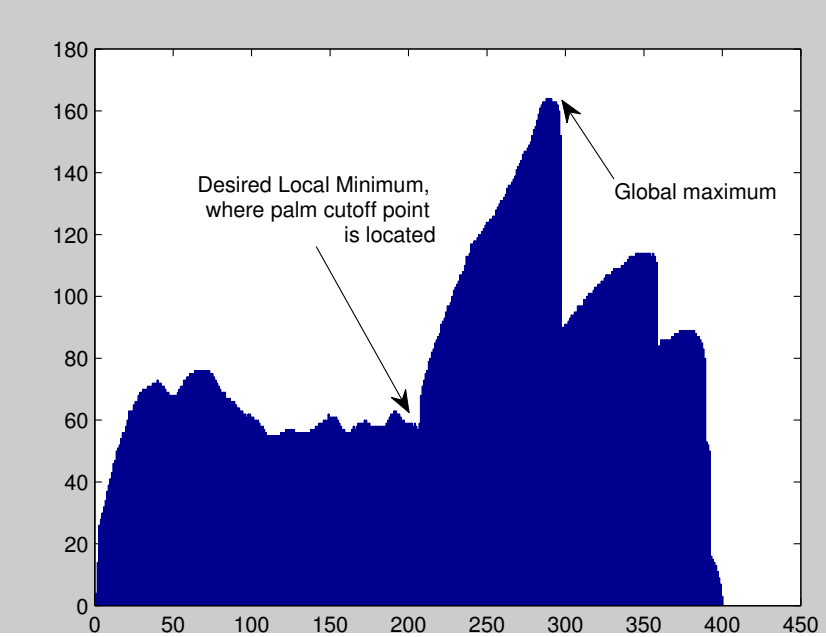


Figure 2.4. This a plot of the sequence of widths. The desired local minimum is highlighted indicating the position for the palm cutoff point.

Weighted Orthogonal Least Squares. The weights are generated from a Gaussian distribution. An implementation of this smoothing technique can be found at the web [23]. To be able to calculate the local regression lines we must define an order for the all pixels x'_i that $\in \mathcal{S}' = \{x'_1, x'_2, \ldots, x'_m\}$. Such an order can be defined with a boundary tracing algorithm. We employ the *Moore-Neighbor-Tracing Algorithm*.

2.3.3 Moore-Neighbor Tracing Algorithm

For a given pixel P_i we can define it's *Moore Neighborhood* as the set M_i of $8-connected$ adjacent pixels, where $M_i = \{P_{i1}, P_{i2}, P_{i3}, P_{i4}, P_{i5}, P_{i6}, P_{i7}, P_{i8}\}$ as seen in Figure 2.6.

Given a binary image that consists of pixels that belong to the same connected component we first need to define the start pixel for our tracing algorithm. We find our start pixel by starting from the leftmost column and visiting pixels from top to down. The first

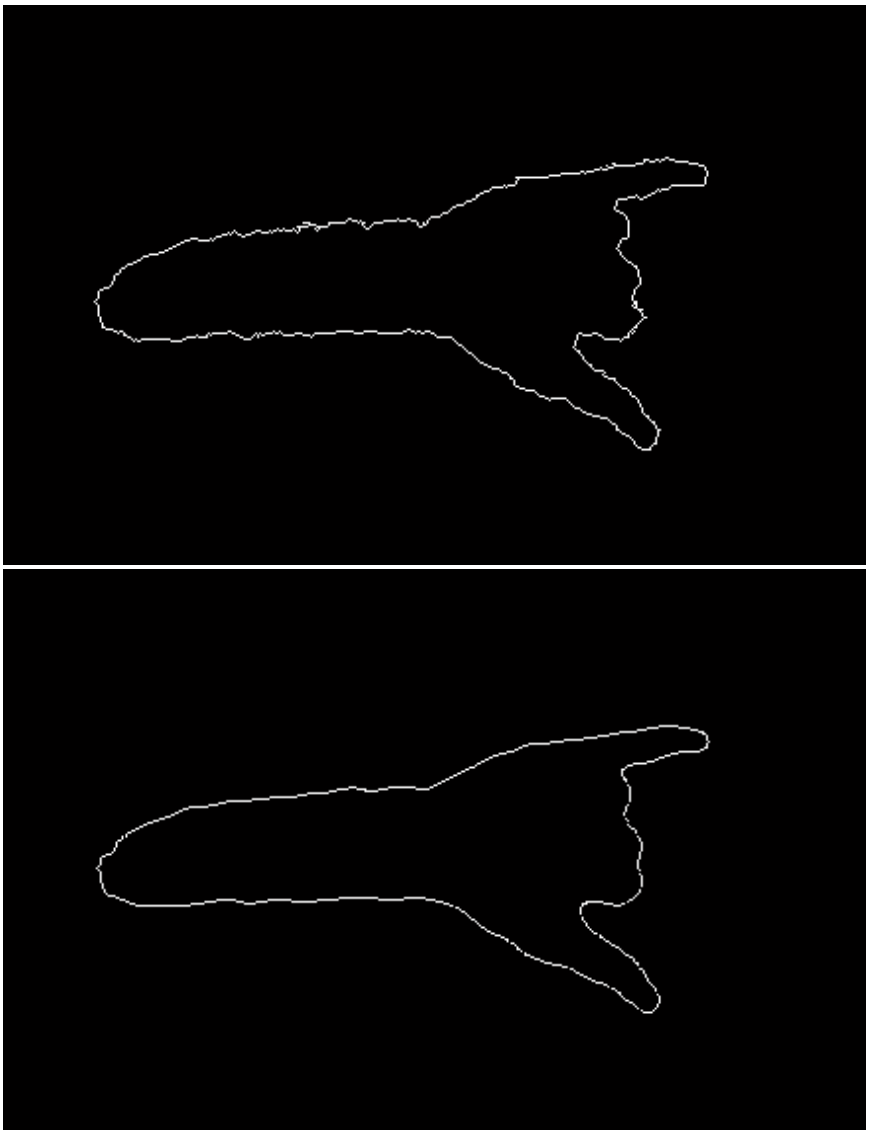


Figure 2.5. At the top image we can see the $2D$ contour before the smoothing operation. At the bottom image we can see the $2D$ contour after the smoothing operation. Original size for both images is 640×480.

non-zero pixel that we encounter is our start pixel. Next, we will extract the contour by going around the pattern in a clock-wise order. The algorithm also works if we choose an anti-clockwise order. We can choose either order however we need to follow the same convention until the algorithm terminates. Every time we visit a non-zero pixel P_i we *back-track*, which means we go back to the zero pixel we were previously standing on. After

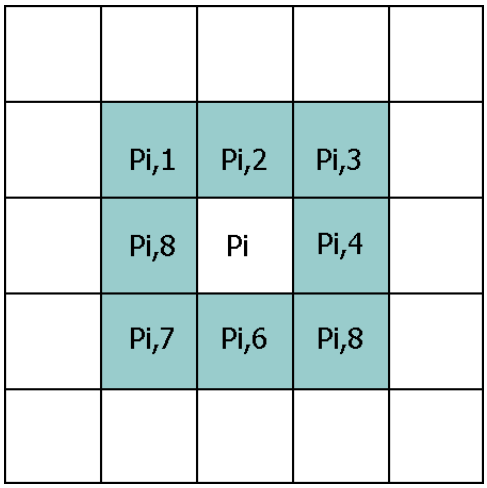


Figure 2.6. This is the *Moore Neighborhood for a given pixel* P_i.

we *backtrack* we go around pixel P_i in a clock-wise (or anti-clockwise) order, visiting the pixels in P_i's *Moore Neighborhood* until we find another non-zero pixel. The termination criterion is to visit the start pixel twice. A more formal description of the *Moore-Neighbor Tracing* technique is presented below at Algorithm 1.

Note that different termination criteria can change the final result of algorithm 1. For more details we refer the reader to [24]. In our experiments the contour tracing method terminates when the "second" pixel in the loop is revisited, entered from the same direction as it was entered on its first visit. A *MATLAB* implementation of the method can be downloaded from [25]. A demonstration of the *Moore-Neighbor Tracing* algorithm can be found in Figure 2.7. The red arrow denotes from which direction we entered the start pixel (i.e., 1). Then we backtrack (blue arrow facing up) and search in the *Moore-Neighborhood* of the start pixel. The first non-zero pixel we encounter is pixel 2. We follow the same procedure until we visit the start pixel for the second time.

In Figure 2.8 we depict the result of the method when applied to some of the arm/hand contours that we use in our experiments.

input : A set of pixels $P = \{p_1, p_2, \ldots p_m\}$, belonging to a 8-connected component

output : A sequence of ordered boundary pixels $O(O_1, O_2, \ldots, O_k)$, $1 \leq k \leq m$

Set O to be empty;

Find the start pixel s, insert s in O;

Set current pixel p to s, $p = s$;

Backtrack and set c to be the next clockwise pixel in $M(p)$;

// c is the current pixel under consideration, i.e., c is in $M(p)$

// $M(p)$ is the *Moore-Neighborhood* of current pixel p

while $(c \neq s)$ **do**

    if c *is non-zero* **then**

        insert c in O;

        set $p = c$;

        backtrack (move the current pixel c to the pixel from which p was entered);

    else

        current pixel c becomes the next clockwise pixel in $M(p)$;

    end

end

Algorithm 1: The Moore-Neighbor Tracing algorithm

2.3.4 Gradient Descent

After the original contour is smoothed the sequence of widths is further filtered with a 1D horizontal mask of ones and of size 5. The original contour smoothing is needed in order to reduce the total number of local minima that could be created due to rugged hand contours. Rugged contours are caused because of the low resolution of the depth data, as can be seen in Figure 2.5. The next step is to perform a gradient descent in order to identify the local minimum at which the palm cutoff point will lie. As a starting point we choose the global maximum (i.e., the highest width) which will always reside in the hand area. Then

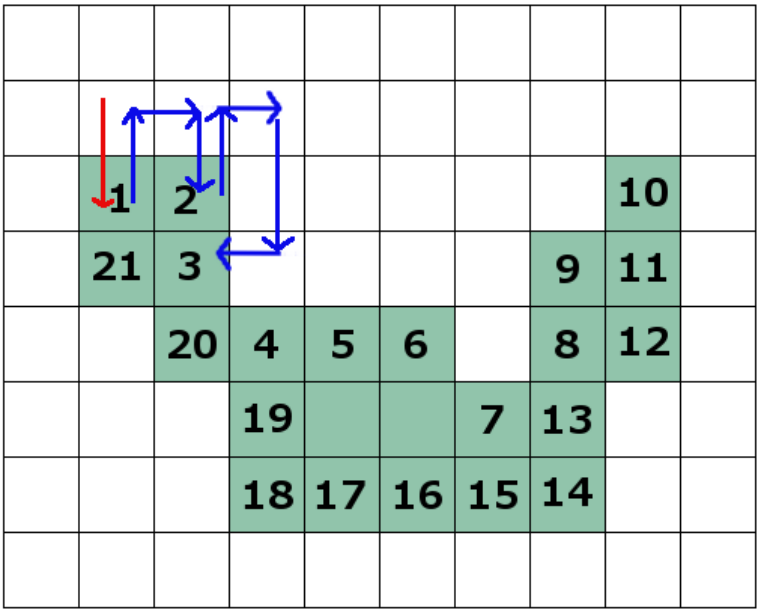


Figure 2.7. A demonstration of the *Moore-Neighbor Tracing* algorithm. Red arrow denotes from which direction we entered the start pixel (i.e., 1). Next blue arrow facing up is *backtracking*. We search in the*Moore-Neighborhood* for the next non-zero pixel (i.e., 2). The algorithm terminates when 1 is visited for the second time.

we move towards the end of the arm until we reach our local minimum. In Figure 2.4 we can see a plot of the sequence of widths along with the desired local minimum where the palm cutoff point is located.

2.4 Framework for Hand Pose Estimation

We model the hand as an articulated object, consisting of 16 links: the palm and 15 links corresponding to finger parts. Each finger has three links (Figure 2.9). There are 15 joints, that have a total of 20 degrees of freedom (DOFs). For the 20-dimensional vector of joint angles we use synonymously the terms “hand shape” and “hand configuration.”

The appearance of a hand shape also depends on the camera parameters. For simplicity, we consider only the camera viewing direction (two DOFs), and image plane orientation. We use the terms “camera parameters,” “viewing parameters” and “3D orienta-

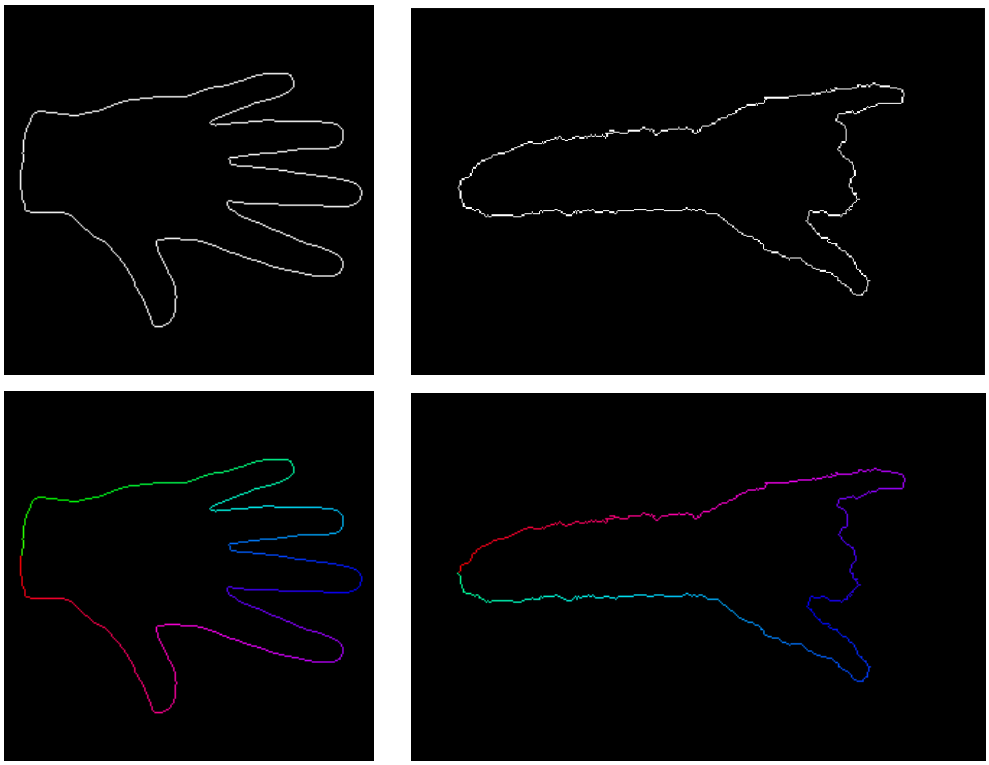


Figure 2.8. On the top row one can see two original edge images from our database. On the bottom row one can see the respective edges images after applying the contour following algorithm (*Moore-Neighbor Tracing*). Order of pixels is denoted with color intensities starting from Red then Magenta, Blue, Cyan, Green, Yellow and Orange. Note that for the bottom left image we have used anti-clockwise order and for the bottom right image clockwise order.

tion" synonymously to denote the three-dimensional vector describing viewing direction and camera orientation. Given a hand configuration vector $C_h = (c_1, ..., c_{20})$ and a viewing parameter vector $V_h = (v_1, v_2, v_3)$, we define the hand pose vector P_h to be the 23-dimensional concatenation of C_h and V_h: $P_h = (c_1, ..., c_{20}, v_1, v_2, v_3)$.

Using these definitions, our framework for hand pose estimation can be summarized as follows:

1. Preprocessing step: create a database containing a uniform sampling of all possible views of the hand shapes that we want to recognize. Label each view with the hand pose parameters that generated it.

Figure 2.9. The articulated hand model. The palm and 15 finger links are shown in different colors.
.

2. Given an input image, retrieve the database views that are the most similar. Use the parameters of the most similar views as estimates for the image. The most similar views (Figure 2.1) are retrieved according to a similarity measure (e.g., Euclidean distance, Chamfer distance)

2.4.1 Database

Our database contains right-hand images of 20 hand shape prototypes (Figure 2.10).

Each prototype is rendered from 84 different viewpoints (Figure 2.11), sampled approximately uniformly from the surface of the viewing sphere.

The rendering is done using *POSER* [1], a $3D$ rendering software package for the posing, animating and rendering of 3D polymesh human figures. *POSER* includes many ready to use $3D$ content items like hands, lights, cameras, materials, scenes etc. We specifically used the hand library that includes hand poses of American Sign Language (ASL).

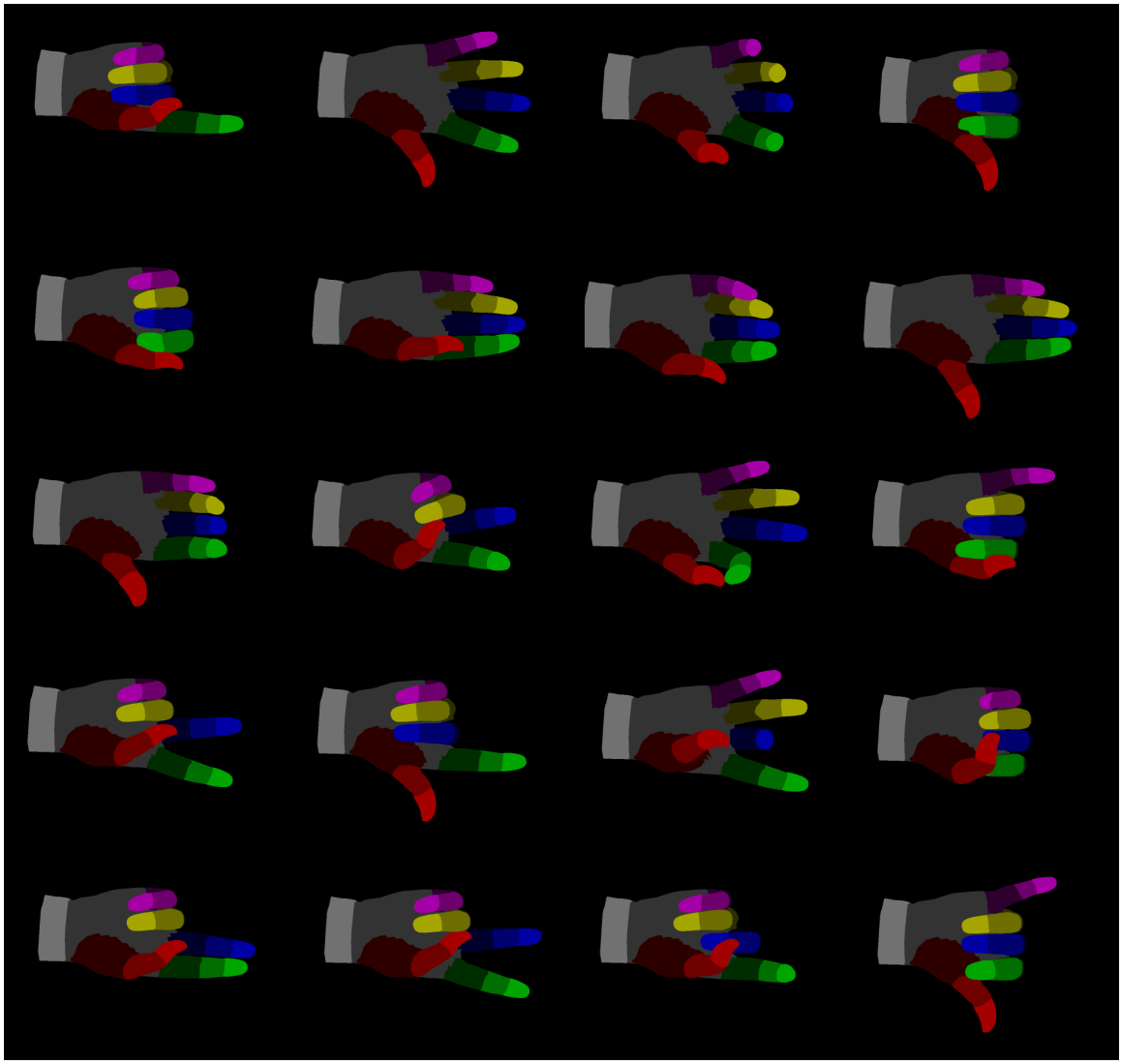


Figure 2.10. The 20 basic shapes used to generate model images in our database. Each basic shape is rendered from 86 different viewpoints.

However the library contains hand poses of various types like hand signals, counting, action poses and gestures. We found this library very useful as it helped us create a vast amount of data for training and testing purposes in our experiments. Collecting such a huge database of real hand images would have been extremely strenuous if not unrealistic.

Given a hand model and the $3D$ rendering computer graphics software we can render images under various lighting conditions and by applying different type of "textures".

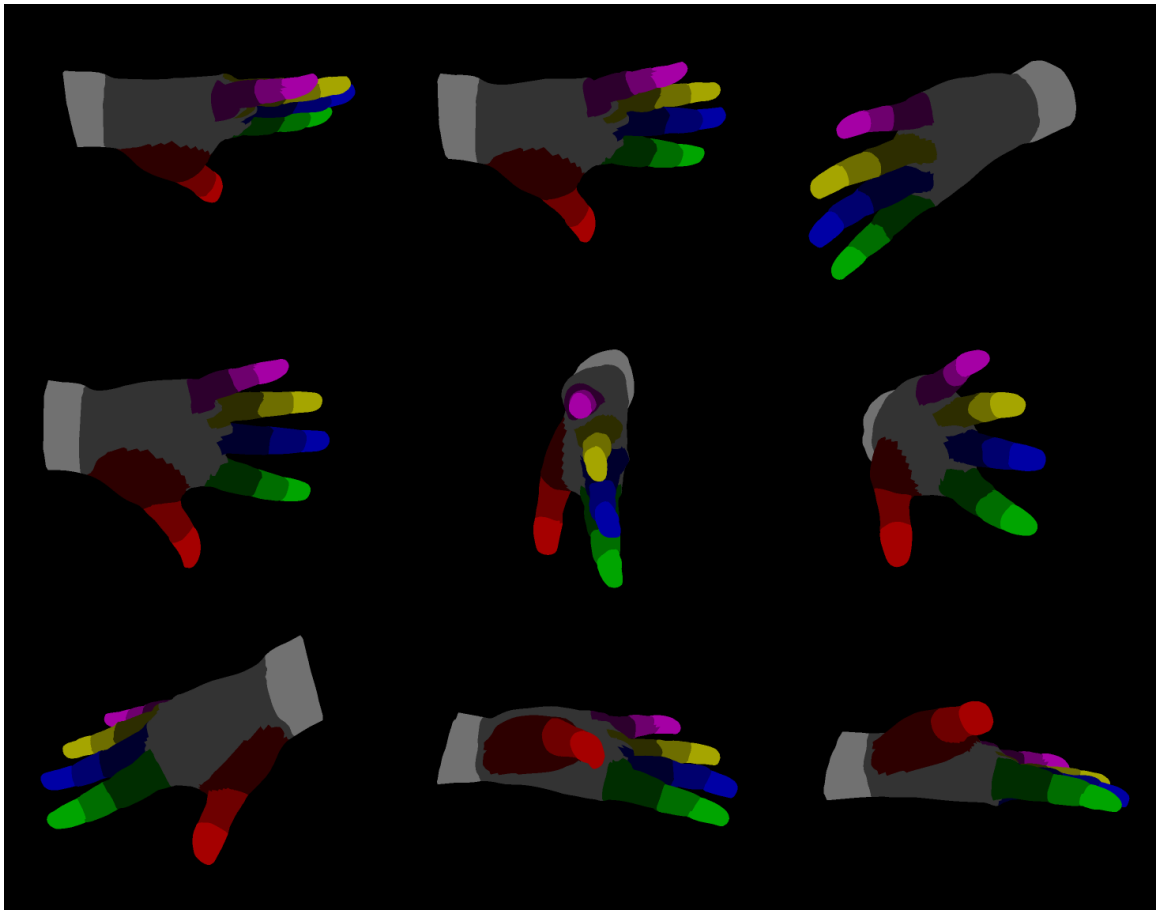


Figure 2.11. Nine $3D$ orientations of the same hand shape.

Depending on our experiments, we can for example render images with skin color texture overlayed on top of the hand model. We can also simulate the effect of shadowing by adding various light sources and appropriate “materials” and “textures”. In Figure 2.12 we can see the same hand model rendered with different settings.

Every $3D$ modeling scene in the *POSER* environment has at least one viewing camera that is fully customizable. We can control the camera’s orientation and position in $3D$ space as well as the focal length etc. For our experiments we consider only the camera viewing direction (two DOFs), and image plane orientation. When rendering our images the $3D$ hand model can be seen as placed at the center of a sphere and the camera moving

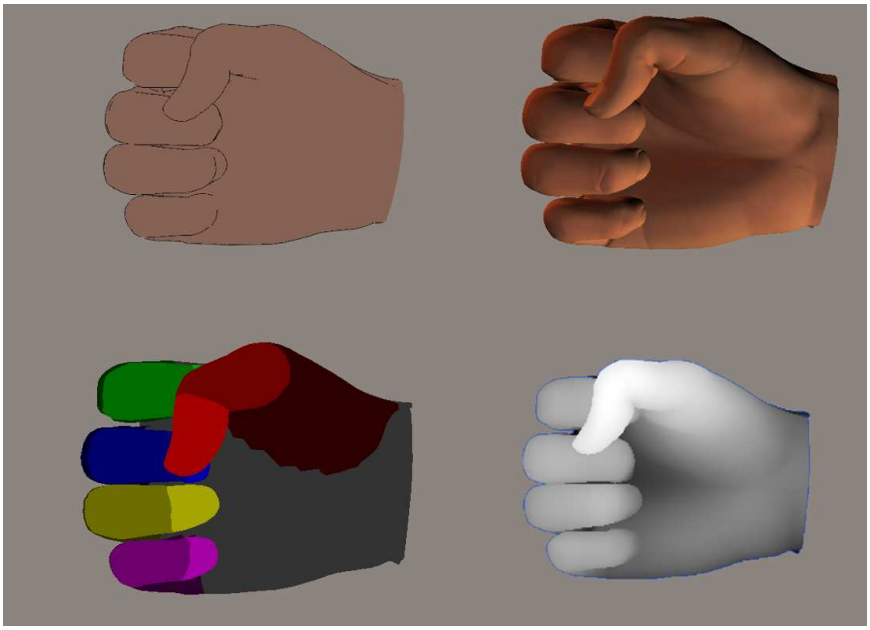


Figure 2.12. Four different type of textures for rendering the same hand shape. On the top left we can see a ”cartoon”-like skin color texture. Top right is a photo-realistic skin color texture. Bottom left all different joints have been rendered with a different color. Bottom right one can see a depthmap, where each pixel intensity encodes the distance from the camera.

along the surface of that sphere always pointing at the center of it. The camera's starting position can be regarded as the "north pole" of the sphere. Any camera placement on the surface of that sphere can be denoted with two parameters: *latitude* and *longitude*. We also use one more parameter for the image plane rotation. In the end, our viewing parameter vector can be modeled as:

$$V_h = (v_1, v_2, v_3) = (latitude, longitude, image\_plane\_rotation) \tag{2.3}$$

Each database image is associated with the viewing parameter vector V_h that has been used during the rendering phase. For example, the image that has been generated when camera is placed at the "north-pole" of the viewing sphere and has no image plane rotation is associated with the viewing parameter vector $V_h = (0, 0, 0)$. In order to create our whole database of training images we sample the whole viewing sphere from 84 differ-

ent viewpoints or pairs of *(latitude, longitude)*. To accommodate rotation-variant similarity measures (like the Chamfer distance), 48 more images are generated from each viewpoint, corresponding to 48 uniformly sampled rotations of the image plane. Overall, the database includes $48 \times 84 = 4032$ views of each hand shape prototype and $4032 \times 20 = 80,640$ images overall. We refer to those images using the terms "database images", "model images", or "synthetic images".

2.4.2 Similarity Measures

2.4.2.1 Chamfer distance

The Chamfer distance [26] is a well-known method to measure the distance between two edge images. Edge images are represented as sets of points, corresponding to edge pixel locations. The X-to-Y directed Chamfer distance $c(X, Y)$ is defined as

$$c(X, Y) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y} \|x - y\| \,, \tag{2.4}$$

where $\|a - b\|$ denotes the Euclidean distance between two pixel locations a and b. The undirected Chamfer distance $C(X, Y)$ is

$$C(X, Y) = c(X, Y) + c(Y, X) \,. \tag{2.5}$$

We will use the abbreviations DCD to stand for "directed Chamfer distance" and UCD for "undirected Chamfer distance."

2.4.2.2 Depth Matching

Test data have been captured via the Kinect™ device which offers two synchronized streams, one RGB and one depth stream. Frame resolution is 640×480. A depth image is a gray-scale image, where each pixel is assigned an intensity value according to how far or close it is located from the camera. We have also managed to create depth-maps for our

synthetically generated database images using a 3D modeling and animation software [1]. Some examples of our depth-maps are depicted in Figure 2.13. Both, model and test depth images are normalized in order to achieve translation invariance for the z-axis. All depth values are in the range from 0 to 1. The depth similarity measure between two images is defined as the total sum of their pixel-wise Euclidean distances. Through the rest of this chapter we will refer to the depth matching similarity measure as $depthSM$.

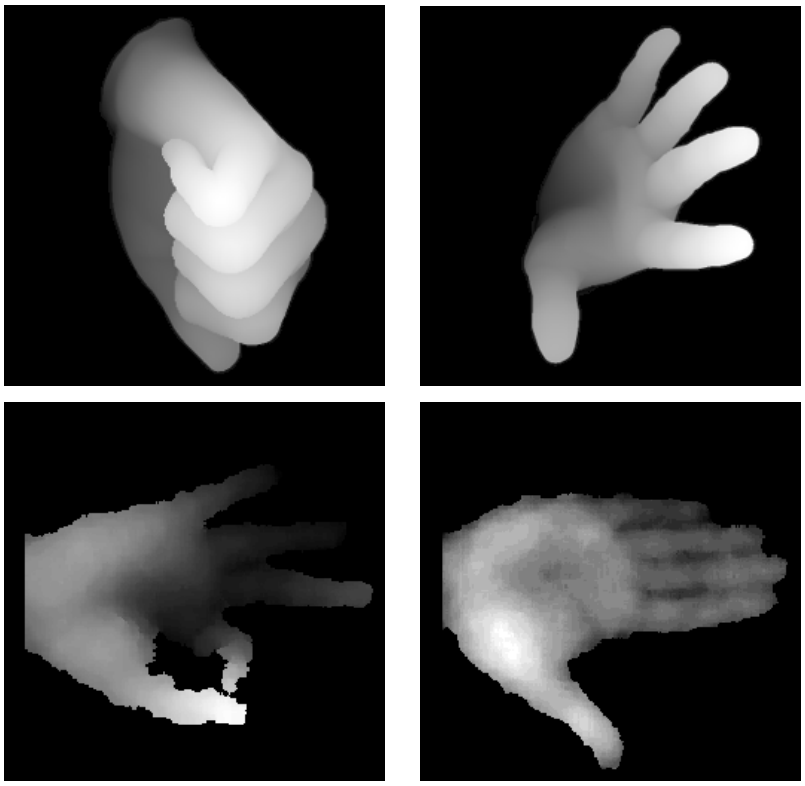


Figure 2.13. The two depth-maps at the left side are "database depth-maps" and have been rendered with a 3D modeling software. The two depth-maps at the right side are "test depth-maps" and they have been captured by the Kinect™ device.

2.4.2.3 Weighted Depth Matching and Chamfer Distance

Another similarity measure is defined by combining the first two similarity measures. In the following equation the $WeightedSM$ is a weighted sum of $depthSM$ and UCD:

$$WeightedSM = l_1 \times depthSM + l_2 \times UCD \tag{2.6}$$

In our experiments we have optimized the weights and demonstrated that the pair $l_1 = 0.8, l_2 = 0.2$ offers the best overall recognition accuracy. In Subsection 2.5.5, Table 2.4, one can see recognition results for a set of different pairs of weights.

2.5 Experiments

2.5.1 Definition of Retrieval Accuracy and Experimental Setup

Towards providing an experimental evaluation the first step is to estimate ground truth for the test data and define *retrieval accuracy*. We have manually established pseudo-ground truth for each test image, by labeling it with the corresponding shape prototype and using the rendering software to find the viewing parameters under which the shape prototype looked the most similar to the test image. This way of estimating viewpoint parameters is not very exact; we found that manual estimates by different people varied by 10-30 degrees. Model views cannot be aligned perfectly because the anthropometric parameters (like finger lengths and widths) of hands in test images do not match those of the model, and because the hand shapes in the real images are not exact replications of the 20 shape prototypes.

After estimating ground truth for test data we need to define what is a *correct match* and also define the *retrieval accuracy*. We consider a database view V to be a *correct match* for a test image I if the shape prototype with which we label I is the one used in generating V, and the manually estimated viewing parameters of I are within 30 degrees of those of

V [27]. On average, there are 30.4 correct matches for each test image in the database. Our measure of *retrieval accuracy* for a given test image I is the rank of the *highest-ranking correct match* that was retrieved for I. 1 is the highest (best) possible rank. In the end we show the percentage of test images for which the highest ranking correct match is within a set of specific ranges (e.g., $1, 1-4, 1-16, 1-32, 1-64, 1-128, 1-256$).

2.5.2 Rendering and Pre-processing Training Images

The original database images have been synthetically generated using a hand model and a computer graphics software [1]. Depending on our experiments and on our similarity measures we use database images with appropriate textures and rendering settings. For example if we are running experiments for the depth similarity measure then our training images are rendered as depthmaps. Another example is when we test Chamfer distance with edge images. In this case we choose to render our training images with a "cartoon"-like texture. Furthermore we have to process these training images and extract contour edges in order to apply the Chamfer distance measure. After we extract edge information a final processing step is required in order to achieve translation and scale invariance for our proposed method. This final step is called *normalization* and is really critical for the performance of our system.

In this short paragraph we will describe how we can achieve *normalization* of our data. Originally the rendered images are of size 1200×942. We can easily segment the foreground from the background since the latter has always the same uniform color. For the set of all foreground pixels we calculate the *Minimum Enclosing Circle (MEC)*. Then we compute the bounding box of the *MEC* and crop the image so as to keep only pixels inside the bounding box. Finally we re-size the cropped image to a fixed size of 256×256.

Our whole pipeline for generating the training database images can be summarized as follows:

- Choose the appropriate texture / material and render all 1680 images (20 handshapes $\times$ 84 viewpoints)
- Further process the images if needed (e.g., Canny Edge Detection)
- Segment foreground pixels and compute their *MEC*
- Crop the image and keep only pixels inside the bounding box of the *MEC*
- Resize the cropped image to a fixed size of 256×256

We need to note that a similar processing pipeline is also required for the given test images. All images need to have the same fixed size 256×256. This way we ensure our system will be translation and scale invariant. In Figure 2.14 we can see a few examples of our normalized database images before and after extracting edges.

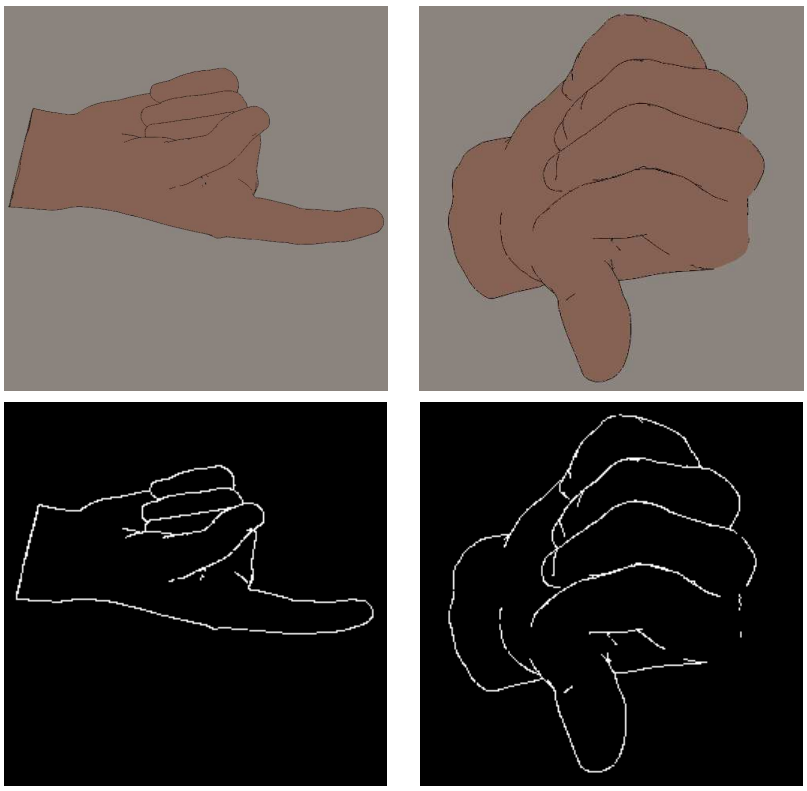


Figure 2.14. At the top row one can see original database images that have been synthetically generated using a hand model and a computer graphics software [1]. At the bottom row we show the respective edge images we are given as input to our method.

2.5.3 Preliminary Results for Datasets with Clean Background

As we further move on to our experimental evaluation the next step is to present some preliminary results and provide the baseline performance of a well known similarity measure for edge images, namely the Chamfer Distance [26]. We refer the reader back to subsection 2.4.2.1 for a more detailed description on Chamfer Distance.

2.5.3.1 Results for the *ASL-RGB* dataset

First we report results for a testing dataset consisting of 174 images representing American Sign Language (ASL) handshapes. Throughout the rest of this dissertation we will refer to this dataset as *ASL-RGB*. The handshapes are real hand images captured in a rather clean background and no clutter is present, as one can see in Figure 2.15).

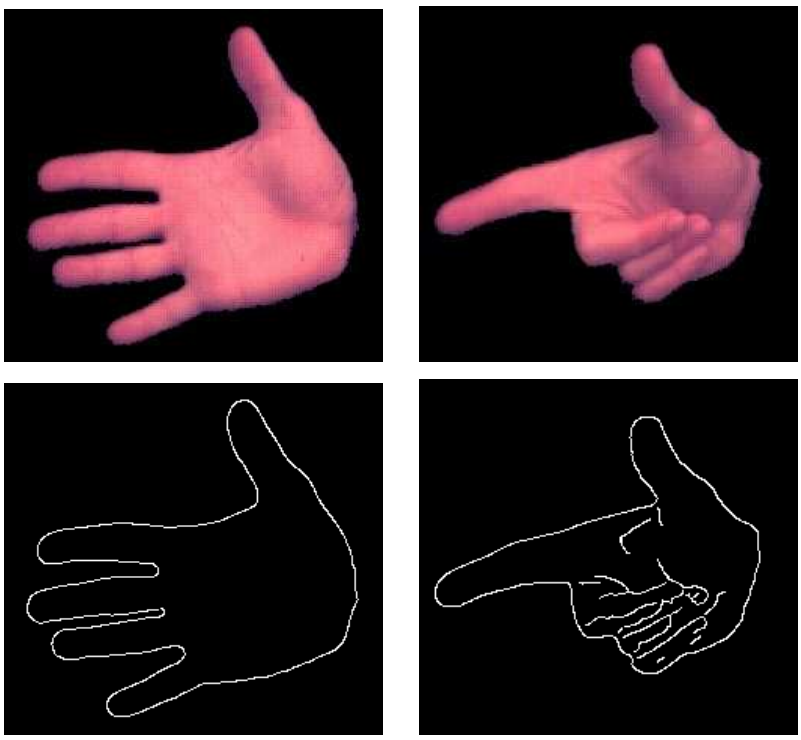


Figure 2.15. Test images from our dataset of 174 ASL handshapes images with clean background. On the top row one can see the original images. Bottom row are the edge images obtained after applying a Canny Edge Detector to the original images.

The training dataset for this first batch of experiments consists of images that have been rendered with the "cartoon"-like texture. Finally we applied Canny edge image detection in order to get the training images with edge pixels (see Figure 2.14). The results for this dataset are presented in Table 2.1.

Table 2.1. Preliminary results for a dataset of 174 ASL handshape images, captured in a rather clean background. For every method used, we show the percentage of test images for which the highest ranking correct match was within each range. DCD stands for "image-to-model directed Chamfer distance". UCD is the undirected Chamfer distance.

Method used	1	1-4	1-16	1-32	1-64	1-128	1-256
DCD	03.45	11.49	16.09	21.26	27.59	38.51	51.72
UCD	14.37	24.71	35.63	43.10	48.85	59.20	71.84

2.5.3.2 Results for the *ASL-Ground-Truth* Dataset

For our second round of experiments we have used the same training images but the testing dataset is slightly different. We have used the ground truth labels from the original test images (i.e., *ASL-RGB*) and generated synthetic images using the corresponding class labels and viewpoint parameters. This new set of test images is very similar to the original dataset as one can see in Figure 2.16. However, it is evident that handshapes with the exact same ground truth labels can have some minor differences that could affect the performance of our hand pose estimation system. This is happening because anthropometric parameters (like finger lengths and widths) vary between a human hand model and a synthetic hand model. Of course the same applies for handshapes that have same ground truth labels but are captured from different users. The purpose for this second round of experiments is to provide an indicative measure of the influence of differences in anthropometric parameters between the test images and training images, see Table 2.2 and Table 2.1. The similarity

measure used here is again the Chamfer distance (directed and undirected). Throughout the rest of this dissertation we will refer to this dataset as *ASL-Ground-Truth*.

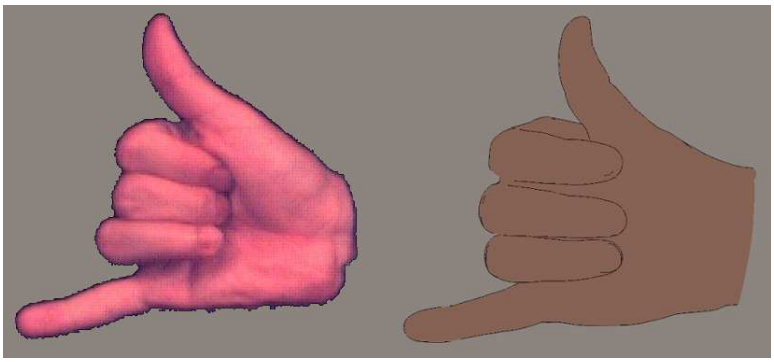


Figure 2.16. To the left one can see the original test image which is a real hand image captured with a clean background. To the right one can see the synthetically generated image by using the same ground truth labels.

Table 2.2. Preliminary results for a dataset of 174 ASL handshape images. For every method used, we show the percentage of test images for which the highest ranking correct match was within each range. $DCDgt$ stands for "image-to-model directed Chamfer distance". $UCDgt$ is the undirected Chamfer distance. Test images have been synthetically generated with a 3D modeling software based on our estimations for ground truth labels of the original test dataset *ASL-RGB* (see Table 2.1)

Method used	1	1-4	1-16	1-32	1-64	1-128	1-256
$DCDgt$	31.03	48.28	63.79	68.97	74.71	81.03	86.78
$UCDgt$	48.28	67.82	77.01	82.18	85.63	91.38	92.53

2.5.4 Preliminary Results for a Dataset with Clutter in the Background

We also report preliminary results (Table 2.3) for a more challenging dataset of 248 ASL handshape images, which have been captured in a cluttered environment (see Figure 2.17). Throughout the rest of this dissertation we will refer to this dataset as *ASL-Clutter*.

The training images remain the same as in subsection 2.5.3. The similarity measure used is Chamfer distance (directed and undirected). The purpose of this round of experiments is to measure the effect of clutter on our proposed system. Clearly, when comparing results between Tables 2.3 and 2.1 we can see a decrease in the recognition accuracy of our system. However we need to mention that the original test images, before segmenting the hand, are of size 340×240. This low resolution could be another factor causing a decrease in the performance.

Table 2.3. Preliminary results for a more challenging dataset of 248 ASL handshape images, which have been captured in a highly cluttered environment. For every method used, we show the percentage of test images for which the highest ranking correct match was within each range. $DCDcf$ stands for "image-to-model directed Chamfer distance". $UCDcf$ is the undirected Chamfer distance

Method used	1	1-4	1-16	1-32	1-64	1-128	1-256
$DCDcf$	04.03	06.45	15.32	2137	27.42	37.10	48.39
$UCDcf$	07.26	15.73	32.66	37.90	46.37	54.03	66.53

2.5.5 Results for the Proposed Depth Similarity Measure

Finally we tested our system on a challenging dataset of 94 right hand images of American Sign Language (ASL) handshapes. The images are captured with the Kinect™ device in a cluttered scene. We will refer to this dataset as *ASL-KinectHandshapes*. These 94 test images are provided in both formats, RGB and depth. When using the RGB test images the training set remains the same as in previous subsections 2.5.4 and 2.5.3. Same applies when testing UCD with *depth contours*. However in this case the test images have been generated from the depth images after extracting the contour pixels of the hand.

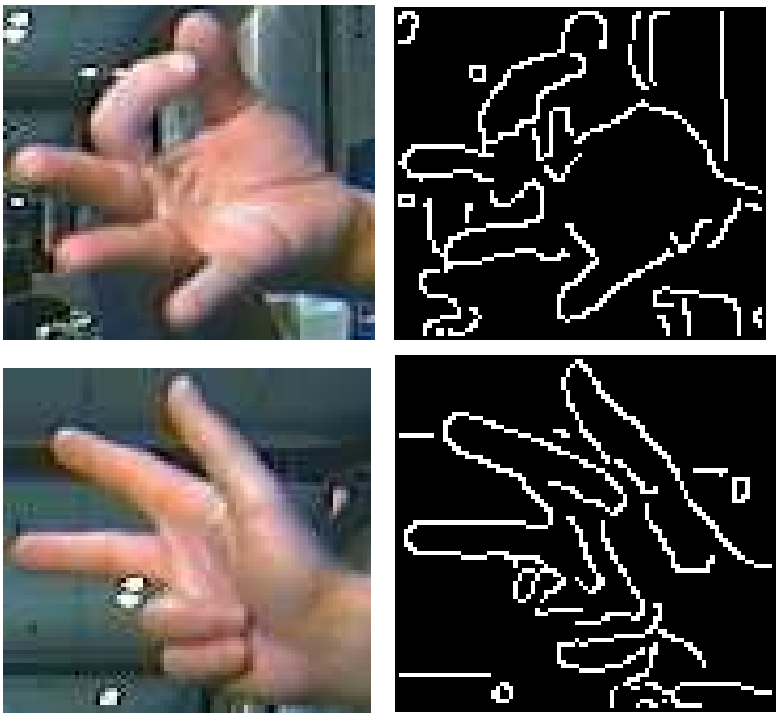


Figure 2.17. Test images from our dataset of 248 ASL handshapes images within a highly cluttered background. To the left one can see the original images and to the right the edge images obtained after applying a Canny Edge Detector to the original images.

For the similarity measure $depthSM$ we employ a different set of training images. This time database images are synthetically generated as depthmaps (see Figure 2.13). Test images are also in the depth format (and not RGB).

Finally for $weightedSM$ similarity measure we use a combination of $depthSM$ and UCD depth contours, using RGB and depth features for training and test images. Table 2.4 reports results for a set of different pairs of weights l_1 and l_2, where l_1 is the weight multiplied with $depthSM$ and l_2 is the weight multiplied with UCD depth contours. $depthSM_{auto}$, contrary to the other similarity measures, uses input images that have been generated using the automatic hand segmentation step.

From the experiments on the *ASL-KinectHandshapes* dataset it is evident that using depth information from the depth-maps enhances the discrimination power of our method,

Table 2.4. Results for *ASL-KinectHandshapes* dataset, using $weightedSM$ similarity measure and a set of different pairs of weights. Optimized recognition rates are reported when $l_1 = 0.8$ and $l_2 = 0.2$.

Weights used	1	1-4	1-16	1-64	1-256
$l_1 = 0.1, l_2 = 0.9$	12.77	23.40	36.17	52.13	71.28
$l_1 = 0.2, l_2 = 0.9$	18.08	26.59	44.68	61.70	77.65
$l_1 = 0.3, l_2 = 0.7$	23.40	37.23	52.12	69.14	79.78
$l_1 = 0.4, l_2 = 0.6$	24.46	42.55	56.38	73.40	82.97
$l_1 = 0.5, l_2 = 0.5$	30.85	42.55	65.59	76.59	85.10
$l_1 = 0.6, l_2 = 0.4$	30.85	44.68	68.08	78.72	86.17
$l_1 = 0.7, l_2 = 0.3$	36.17	50.00	69.14	80.85	86.17
$l_1 = 0.8, l_2 = 0.2$	37.23	50.00	70.21	80.85	88.29
$l_1 = 0.9, l_2 = 0.1$	34.04	48.93	68.08	79.78	87.23

Table 2.5. Results for *ASL-KinectHandshapes* dataset. For every method used, we show the percentage of test images for which the highest ranking correct match was within each range. UCD depth contours, is undirected Chamfer distance between hand contours from depth images and full edges from model images. In UCD color edges, skin color segmentation has been employed to extract the full edges. $depthSM$ is our depth similarity measure, with manual hand segmentation. $depthSM_{auto}$ is our depth similarity measure, with automatic hand segmentation. $weightedSM$ similarity measure has weights $l_1 = 0.8$ and $l_2 = 02$.

Method used	1	1-4	1-16	1-64	1-256
UCD depth contours	14.74	20.0	32.63	40.0	62.11
UCD color edges	9.57	14.89	21.28	32.98	55.32
$depthSM_{auto}$	25.53	38.30	55.32	70.21	84.04
$depthSM$	34.04	44.68	61.70	76.60	87.23
$weightedSM$	37.23	50.00	70.21	80.85	88.29

even in uncontrolled and cluttered environments. We notice that even when we only use contours from depthmaps (i.e., UCD depth contours) it outperforms the case "UCD color edges" where full edges of the hand are available due to color information. Best performance is reported when using the $weightedSM$ similarity measure.

2.5.6 Consolidated Results

Finally, in Table 2.6 one can see results from the previous subsections consolidated. Additionally, we report results for another state of the art method presented in [4]. The authors of that method used for their experiments a testing dataset of 250 real images of right hands. Since our proposed method is tested on very similar datasets (i.e., real hand images in cluttered backgrounds) we believe that all comparisons in this dissertation are fair and meaningful. The best overall performance, when testing against real hand images, is achieved when using the proposed $weightedSM$ similarity measure with weights $l_1 = 0.8$ and $l_2 = 0.2$. Total processing time varies between different implementations of our proposed framework. We provide an indicative measure of the computation time of our method (using $depthSM$) which is about 33 seconds per input image, not including hand segmentation, on a PC with a 2.00 GHz Intel(R) Xeon(R) E5406 processor. The code is rather unoptimized and implemented in MATLAB R2012a.

Table 2.6. Consolidated results. Best overall performance is achieved when using the proposed $weightedSM$ similarity measure with weights $l_1 = 0.8$ and $l_2 = 0.2$. Our proposed method outperforms the state of the art method presented in [4].

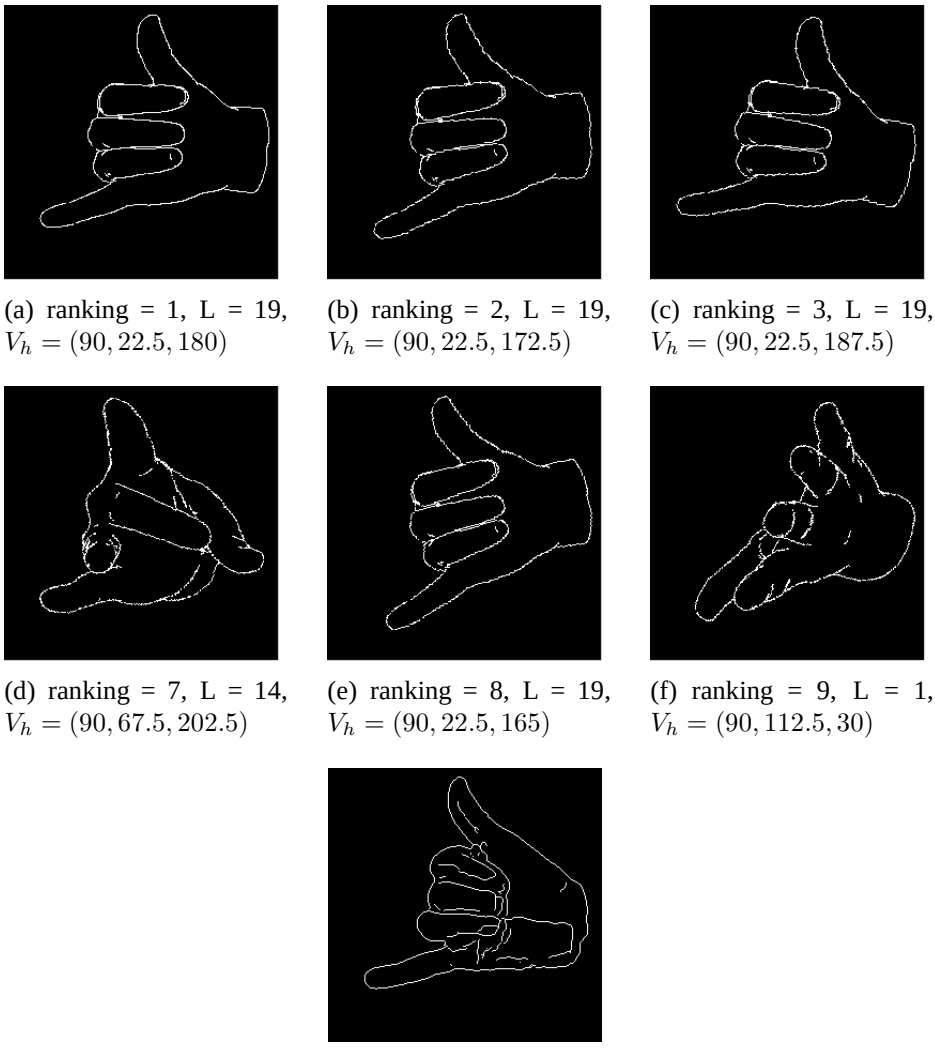
Method used	1	1-4	1-16	1-64	1-256
DCD	03.45	11.49	16.09	27.59	51.72
UCD	14.37	24.71	35.63	48.85	71.84
$DCDcf$	04.03	06.45	15.32	27.42	48.39
$UCDcf$	07.26	15.73	32.66	46.37	66.53
UCD depth contours	14.74	20.0	32.63	40.0	62.11
UCD color edges	9.57	14.89	21.28	32.98	55.32
$depthSM_{auto}$	25.53	38.30	55.32	70.21	84.04
$depthSM$	34.04	44.68	61.70	76.60	87.23
$weightedSM$	37.23	50.00	70.21	80.85	88.29
method from [4]	13.60	26.40	45.20	67.60	84.0

2.5.7 Qualitative Evaluation

In this subsection we will provide a short qualitative evaluation for our similarity measures and some of the experimental results. We will focus on two different cases of query images from the dataset *ASL-RGB*. For each query image we will demonstrate some of the highest ranking database images that have been identified by our method based on Chamfer distance. We need to stress that these highest ranking top matches don't necessarily have to be correct matches even though that is our goal. We would like to remind to the reader that we consider a database view *V* to be a correct match for a test (or query) image *I* if the shape prototype with which we label *I* is the one used in generating *V* , and the manually estimated viewing parameters of *I* are within 30 degrees of those of *V*. For each of the images in the following Figures 2.18 and 2.19 we provide the class label L and the viewing parameter vector $V_h = (latitude, longitude, image\_plane\_rotation)$. First in subsection 2.5.7.1 we demonstrate some of the retrieved model images where the rank of the highest-ranking correct match is 1. Then in subsection 2.5.7.2 we demonstrate some of the retrieved model images where the rank of the highest-ranking correct match is 5601.

2.5.7.1 Retrieved Model Images where Rank of the Highest-ranking Correct Match is 1

In Figure 2.18 we depict a query image that has been given as input to our hand pose estimation system and some of the retrieved model images based on our method. The rank of the highest-ranking correct match is 1, which is depicted in sub-figure 2.18(a). In sub-figures 2.18(b), 2.18(c) and 2.18(e) we can see some more correct-matches with respective ranks 2, 3 and 8. In sub-figures 2.18(d) and 2.18(f) one can see some retrieved model images that are not correct-matches since they have different class labels than the query image. However their rank is extremely high (7 and 9) since our method has falsely regarded them as a very good match for the query image.



(a) ranking = 1, L = 19, $V_h = (90, 22.5, 180)$

(b) ranking = 2, L = 19, $V_h = (90, 22.5, 172.5)$

(c) ranking = 3, L = 19, $V_h = (90, 22.5, 187.5)$

(d) ranking = 7, L = 14, $V_h = (90, 67.5, 202.5)$

(e) ranking = 8, L = 19, $V_h = (90, 22.5, 165)$

(f) ranking = 9, L = 1, $V_h = (90, 112.5, 30)$

(g) Query image, with L = 19 and $V_h = (99, 1, 184)$

Figure 2.18. Query image and some of the retrieved model images.

2.5.7.2 Retrieved Model Images where the Rank of the Highest-ranking Correct Match is 5601

In this subsection and in Figure 2.19 we depict another query image that has been given as input to our hand pose estimation system and some of the retrieved model images based on our method. However in this case our method failed to successfully identify a correct match, since the rank of the highest-ranking correct match is 5601. In sub-figures

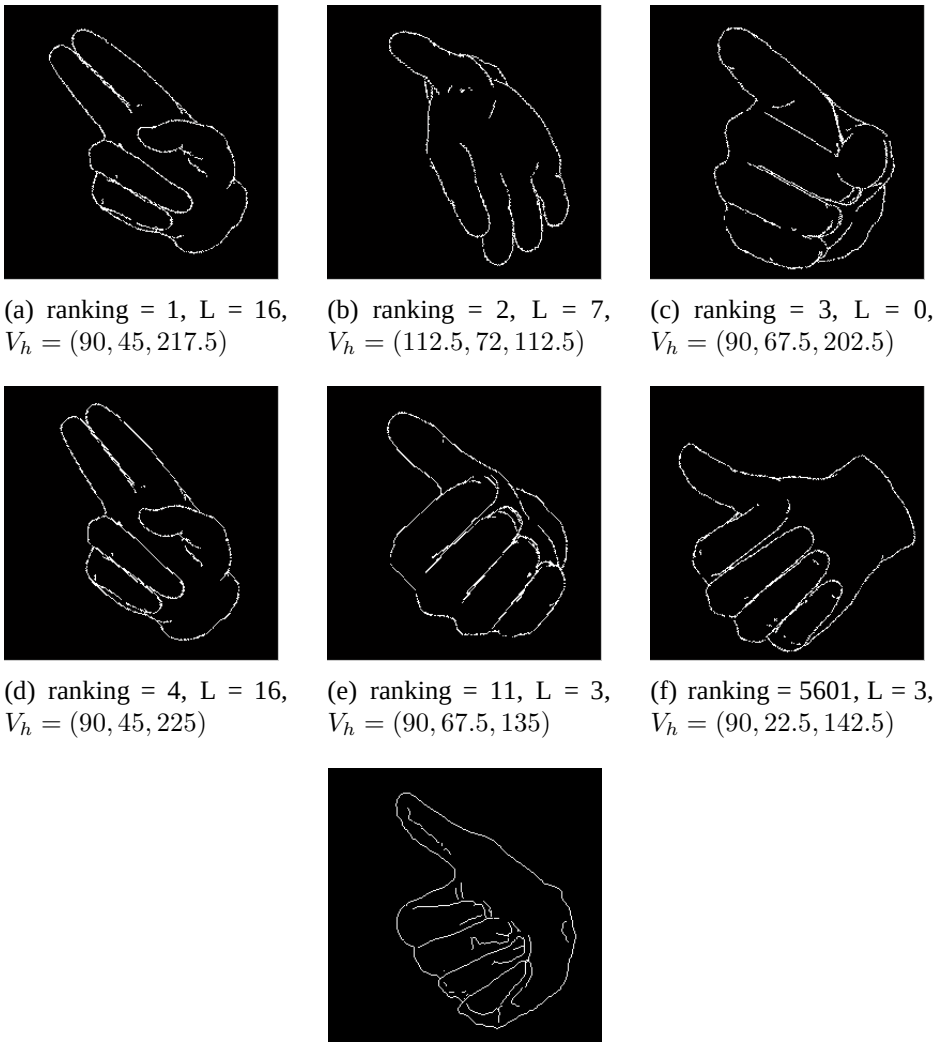
2.19(a), 2.19(b), 2.19(c) and 2.19(d) we can the four highest ranking retrieved images. However none of the is a correct match since their class labels are different than the query class label. In sub-figure 2.19(e) one can see the model image with the highest ranking (i.e., 11) that has a correct class label. However the angle difference between the query and model viewing parameter vectors ($V_{query} = (97, 3, 158)$ and $V_{model} = (90, 67.5, 135)$ respectively) is $diff(V_{query}, V_{model}) = 67.0389$. Since $67.0389 > 30$ we can not classify this retrieved image as a correct match. In sub-figure 2.19(f) we can see the correct match with the highest ranking which is 5601. Clearly our method failed dramatically to identify a correct match. An interesting observation for the database images depicted in sub-figures 2.19(e) and 2.19(f) is that visually they look very similar. When comparing the latter with the query image (see sub-figure 2.19(g)) we notice as main difference the edge pixels close to the carpus. This indicates that the segmentation of the hand is a critical factor for the successful performance of our system.

2.5.8 Results for Automatic Hand Segmentation Method

We have also evaluated quantitatively our automatic hand segmentation method. To this end we compared for all of our test images the distance between the automatically generated *palm cut-off points* and the manually specified ones. For the 80.85% of our images the distance (measured in pixels) is negligible. The distance is less than 15 pixels for the 85.11% and less than 25 for the 92.55% of the images (Table 2.7).

Table 2.7. Results for our Hand segmentation method

range (measured in pixels)	0	0-15	0-25	0-45	0-68
percentage of test images	80.85	85.11	92.55	96.81	100



(a) ranking = 1, L = 16, $V_h = (90, 45, 217.5)$

(b) ranking = 2, L = 7, $V_h = (112.5, 72, 112.5)$

(c) ranking = 3, L = 0, $V_h = (90, 67.5, 202.5)$

(d) ranking = 4, L = 16, $V_h = (90, 45, 225)$

(e) ranking = 11, L = 3, $V_h = (90, 67.5, 135)$

(f) ranking = 5601, L = 3, $V_h = (90, 22.5, 142.5)$

(g) Query image, with L = 3, and $V_h = (97, 3, 158)$

Figure 2.19. top 10 results.

Total processing time for the segmentation algorithm, on a PC with a 2.00 GHz Intel(R) Xeon(R) E5406 processor, is about 10 seconds. The code is rather unoptimized and implemented in MATLAB R2012a.

2.6 Conclusions and Future Work

We have presented a new method for hand pose estimation from a single depth image. Our method combines a novel hand segmentation method and a similarity measure ($depthSM$) based on depth information from depth images.The similarity measure is used to retrieve the best matches from an image database, thus providing estimates for the 3D hand pose and hand configuration. Depth information increases the discrimination power of our method, according to the experiments conducted. At the same time, differences in anthropometric parameters and clutter in background are two important factors influencing recognition accuracy of our system. Experimental evaluation of these two factors has been provided by quantitatively measuring their influence on the performance of our proposed similarity measures. Overall, retrieval accuracy is still too low for the system to be used as a stand-alone module for 3D hand pose estimation. However estimating hand pose from a single image can be useful in automatically initializing hand trackers. Our system currently doesn't achieve real-time performance. In order to do so and since our method is inherently parallel, we are planning to take advantage of the GPU's processing power. Additional future work will be to define more sophisticated similarity measures further exploiting depth information. At this point depth information and the way we have used it can be regarded as $2.5D$. By using the Kinect™ camera's intrinsic and extrinsic parameters we can construct $3D$ point clouds and start exploiting this richer source of information. We could experiment with features like surface normals and other $3D$ feature descriptors. All the aforementioned future directions address interesting research problems but implementing them is out of the scope of the current dissertation.

CHAPTER 3

HAND TRACKING USING DEPTH DATA

3.1 Introduction

Accurate detection and localization of moving hands in unconstrained environments without any elaborate experimental setup remains a challenging task. An accurate and robust hand tracking can be applied in sign language recognition, human-computer interfaces and virtual reality environments. There is an extensive literature on hand detection/ tracking and part of it has been presented at previous Chapter 2. In this chapter we focus on a hand detection/tracking module that is integrated within a more general framework for gesture recognition, which is presented at Chapter 4. The main differences of our hand tracking method presented here with respect to the hand detection method presented in the previous Chapter 2 are the following:

1. Our hand tracking method assumes temporal continuity and hence employs motion cues. We use additional information from multiple frames (i.e. previous, current and next) as opposed to relying on a single frame (i.e. current frame).
2. Here we are not interested in articulated hand tracking. We are interested at locating the hand but we don't care about the details of motion of each finger and the palm altogether.
3. Our hand detection module is integrated into a more general framework for gesture recognition, which is presented at Chapter 4.

Our hand detector is based on motion from frame differencing which we combine with a depth segmentation according to the depth information we have for each pixel. Our detector can return a single best hand location or a list of the best candidate hand locations.

We evaluate our method on a Microsoft Kinect™ based video dataset of American Sign Language (ASL) signs. This dataset has been designed for body part detection and tracking research as well as for ASL sign recognition. At the same time we also report results on the performance of a popular skeleton tracker which is provided by the not-for-profit OpenNI consortium [28]. In this way we establish a benchmark for the future assessment of more advanced detection and tracking methods that utilize RGB-D data. The contributions of this chapter can be summarized as follows:

1. A new hand tracking method based on RGB-D information that is an integral part of the gesture recognition system proposed later in this dissertation (see Chapter 4)
2. A structured motion dataset of American Sign Language (ASL) signs captured in RGB and depth format using a Microsoft Kinect™ sensor, that will enable researchers to explore body part (i.e. hands) detection and tracking methods, as well as gesture recognition algorithms.
3. Experimental evaluation of our method against a popular skeleton-tracker that could be used as a baseline performance when evaluating new state of the art tracking methods.

In the next section 3.2 we describe in detail our new hand tracking method which is based on depth data. Then in section 3.3 we introduce our new RGB-D dataset of ASL signs. Experimental setup and results are presented in section 3.4. Finally we conclude and discuss future work at section 3.6.

3.2 Methodology

In this subsection we describe our method in more detail. Each pixel in an image taken from a Kinect™ camera is assigned a value according to how close or far it is from the plane defined by the camera lenses. For depth video, our proposed detector is based

Figure 3.1. Hand detection in depth images: original image (top left), depth image (top right), segmentation using depth (middle left), the connected component corresponding to the gesturing human (middle right), scores based on multiplication of frame differencing and depth (bottom left), single top candidate (bottom right). One can see that the depth detector here successfully detects the hand of the gesturing person.

on motion and depth information and is applied on depth images captured from a Kinect^{TM} camera. The depth images are grayscale images with values ranging from [0-2046] and value 2047 denotes an invalid depth pixel. First we find the connected component of the person performing the ASL sign . To achieve this we calculate for every pixel the absolute value of the difference, in depth, between that pixel and the top, bottom, left and right neighboring pixels. If the absolute value is over a threshold (10 in our experiments) the pixel is zero otherwise one. In this way we create a binary image where edges are depicted on areas with abrupt changes in depth intensity. Now we have a cleanly segmented image compromising of connected components that have the following property: within each component the depth intensity values of neighboring pixels never increase over 10. The next step is to calculate the average depth for the 5 biggest connected components. The component with the lowest mean depth value is assumed to be the person performing the sign. Now that we know which pixels belong to the person we calculate their max and median depth values.

Next, we calculate a score matrix based on frame differencing. Frame differencing is the operation of computing, for every pixel, the minimum of two values; the absolute value of the difference in intensity between the current frame and the previous frame and the absolute value of the difference in intensity between the current and the next frame. After frame differencing we compute another score matrix which is our depth image minus the median depth value for the person. We multiply element by element those two matrices and we apply a mask over the final matrix. The mask has the following properties:

- all invalid pixels in the previous, current and next frame are zero. The reason we do this is because the captured depth images have a lot of noise which can be regarded as ”motion”.
- all pixels in the motion score matrix that have a value lower than 1 are zero
- all pixels that their intensity value is over the max depth value are zero.

After we have applied the mask on our score matrix we use a vertical 1D filter of ones with some predefined size and finally on the new matrix we apply the same filter horizontally. The min value of our final score matrix denotes where our hand is located. Figure 3.1 illustrates examples of input, output, and intermediate steps for this detector.

We evaluate our method on a dataset of ASL signs which is described in the following section 3.3

3.3 Description of ASL Dataset

Our goal is to create a structured motion dataset that will enable researchers to explore body part (i.e. hands) detection and tracking methods, as well as gesture recognition algorithms not possible with such datasets as the ASLLVD [29] by including scene depth information. The dataset is being recorded with a Microsoft Kinect™, which allows us to capture both color video and frames that include scene depth information. Figure 3.2 shows an example from one of the recording sessions. In this particular representation, the darker gray areas of the image are located closer to the camera. The black regions are portions of the scene for which depth information was not available.

3.3.1 Discussion of Related Gesture Recognition Datasets

One of the highest quality video datasets useful for gesture recognition research is the American Sign Language Lexicon Video Dataset (ASLLVD) [29]. It consists of a large set of recordings from multiple camera angles of the signs contained in the Gallaudet Dictionary of American Sign Language [30], performed by native signers. Each sign is annotated with the gloss label (approximate English translation), start and end frames, hand shapes at the start and end frames, and position of the hands and face, with multiple examples per sign. Such datasets, while extremely useful, lack any information about scene depth, since they were recorded with standard color video cameras. Thus, when using them, researchers

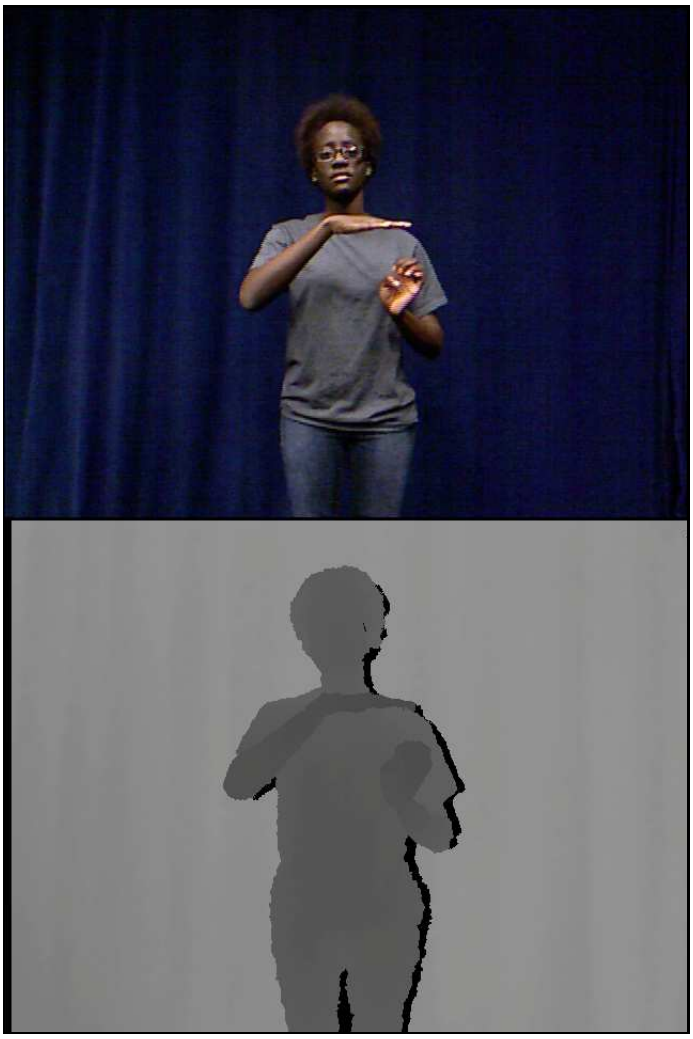


Figure 3.2. Sample dataset sign frame. Top: color video frame; Bottom: depth video frame.

suffer from the limitations of having to use conventional 2D hand detection and tracking algorithms. Hand tracking using standard video is particularly challenging because of occlusions, shading variations, and the high dimensionality of the motion.

Guyon, et al., present a 3D gesture dataset in [31] containing 50,000 gestures recorded with the Microsoft Kinect by 20 different users that is organized into 500 batches of 100 gestures. There are several limitations of this dataset, however. First, the video is recorded at only 10 frames per second and at a resolution of 320×240. Secondly, only 400 frames

are manually annotated with any skeletal information, which makes it difficult to quantify the efficacy of any body part tracker being developed. Finally, as the video is offered only as AVI files, we cannot translate the pixels into x,y,z coordinates in a 3D world reference frame. The dataset we are developing addresses these limitations by using 640×480 resolution at 25 frames per second and providing access to the raw scene depth information.

3.3.2 Size and Scope

We hope that our final dataset will contain most of the 3,000 signs found in The Gallaudet Dictionary of American Sign Language [30], which will offer an abundance of complex movements of the hands and arms. Currently, 1113 signs—both one-handed and two-handed—have been recorded with one fluent signer and 200 with another fluent signer, but in the future, we will add more signers, so that there are multiple examples of each sign. A recent published work which offers a more detailed description of the proposed ASL dataset can be found in [32].

As with [29], finger-spelled signs, loan signs, and classifiers are not included in the dataset. A finger-spelled sign is a word that is spelled out by using the manual alphabet. When a signer has to use a letter that is part of the overall sign, that letter is known as a *loan sign*. Classifiers provide additional information about the object being signed, but since there are infinite variations of them, they are excluded. The ASLLVD paper [29] contains more information about the motivations for excluding certain types of signs.

3.3.3 Technical Specifications

Both the Kinect™ color frames and depth frames have a resolution of 640 x 480 pixels and are recorded at frame rate of 25 frames per second. The signers perform groups of ten signs per video in front of a neutral backdrop in a lab with consistent lighting. The signs are performed while standing, and the scene is framed so as to include the region

from about the knees to a few inches above the signer's head. Each video begins with a calibration pose that can be used to detect the signer and initialize tracking. After the pose, between each sign, and after the last sign, the signer returns her hands to her side, creating a clear separation of the signs in the video.

We currently use the OpenNI framework [28] to record the signs in the ONI format, but we may rerecord them in the future with the Microsoft Kinect™ SDK [33] so that researchers can experiment with both platforms. OpenNI is an open source sensing development framework used in many third party APIs. Its purpose is to standardize compatibility and interoperability of Natural Interactive devices and applications. It and third party software developed around it are useful to researchers that want to develop their own detection and tracking tools. Compressed and uncompressed AVIs of the videos are also available.

3.3.4 Annotations

Each video in the dataset is annotated with the start and end frames of each sign so that any sign can be quickly accessed. The first depth video frame of each sign is annotated with a bounding box around the signer's face to give an idea of the scale of the individual in the video. With this information, the researcher has an idea of how to scale the query sign to which it is being compared. Furthermore, every depth frame belonging to a sign is also annotated with bounding boxes around the hands, which give an indication of the hand's location when the box's centroid is calculated. In the future, we will annotate the color video in the same manner.

For potential use in future gesture and sign language recognition research, the annotations also include information about the signs themselves, such as signer ID, file locations, sign type (two-handed/one-handed), and gloss, or rough English translation. The hand and

face annotations for an example sign frame are shown overlain on the depth frame image in figure 3.3.

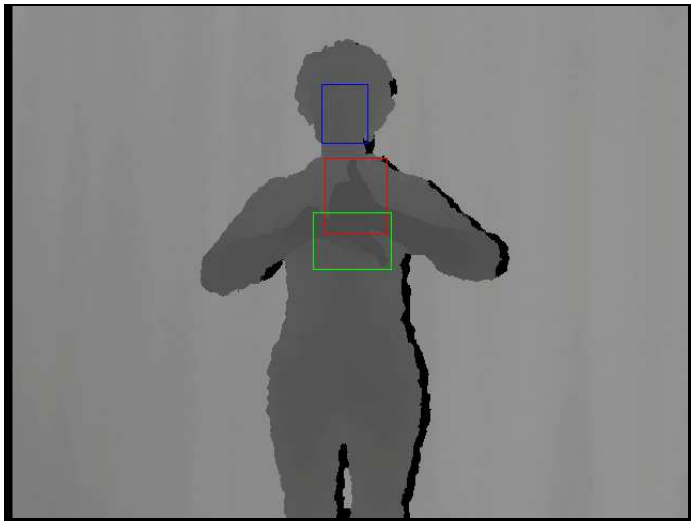


Figure 3.3. Sample hands and face annotations of a single depth video frame.

3.4 Experimental Setup for the Evaluation of Hand Tracking Methods

In order to establish the benchmark, we chose to use the hand location capabilities of the user skeleton tracker included in the OpenNI 1.5 NiUserTracker sample program [28], since it is freely available to anyone. Once it has found the signer via the standard psi calibration pose, the program creates a skeletal model of the person and tracks joint position movement. In particular, we were interested in the arms and defined the hand locations to be the hand endpoints of the elbow-hand portions of the arms.

To evaluate the efficacy of using the skeleton tracker to approximate the positions of the hands, we used 35 one-handed randomly selected signs from the dataset described in section 3.3 and processed them with the tracker. For the one-handed signs, only the signing hand was considered. Once the hand positions were obtained, they were compared to

the ground truth positions from the manual annotations, and the pixel Euclidean distance between them was recorded as a score, so that a lower score would indicate a closer estimation of the hand's actual location. This operation was performed on each frame of the signs, and the accuracy was calculated to serve as the benchmark for the evaluation of future methods. We have also processed the one-handed signs with the proposed single hand locator (described in Section 3.2) and calculated the results using the same pixel Euclidean distance similarity measure.

3.5 Results for Hand Tracking

We calculated overall accuracy as a percentage of frames in which the automatically generated hand locations fell within in various pixel distances (termed pixel error) of the manual hand annotations. Figure 3.4 shows the accuracy of the OpenNI skeletal tracker and our depth hand locator on one-handed signs. For the skeletal tracker 90% of the frames have a pixel error of about 24 pixels or less. It can also be seen that our depth hand locator does not perform as well on this dataset. It is understandable when we consider that it was written for use in simple hand gestures in which the hand will likely be the closest part of the body to the camera. Indeed, after examination of the results, we determined that it tends to fail when other body parts that are also in movement, such as the elbow, are closer to the camera.

We also calculated the maximum pixel error for each sign. Figure 3.5 shows the results for the skeletal tracker and its comparison to our depth hand detector. We can see that 50% of the signs had a maximum pixel error of about 22 pixels or less when our depth hand tracker [2] was used to detect hands.

Figure 3.6 shows a visualization of three levels of skeleton tracker accuracy from good to poor on a single-handed sign, with the pixel error ranging from a few pixels to a

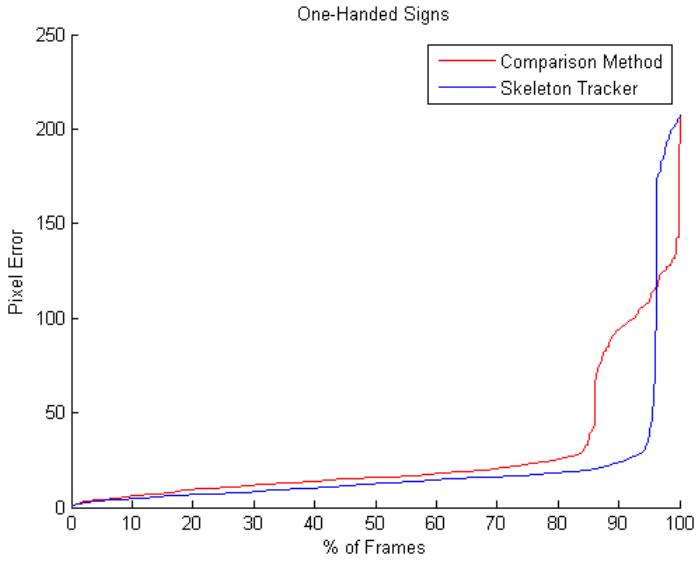


Figure 3.4. Comparison of the skeletal tracker and our method from [2] on one-handed signs.

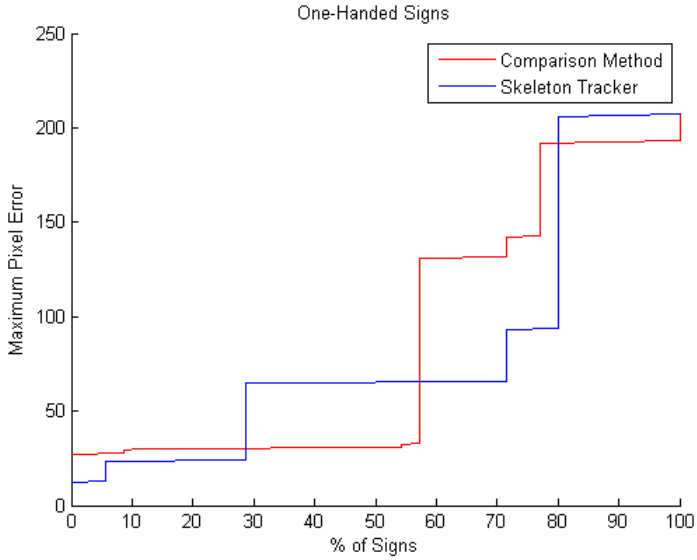


Figure 3.5. Skeletal tracker and depth hand tracking method maximum pixel error on a per sign basis.

few hundred pixels. The manual annotations are shown in green and the skeleton tracker hand locations in red.

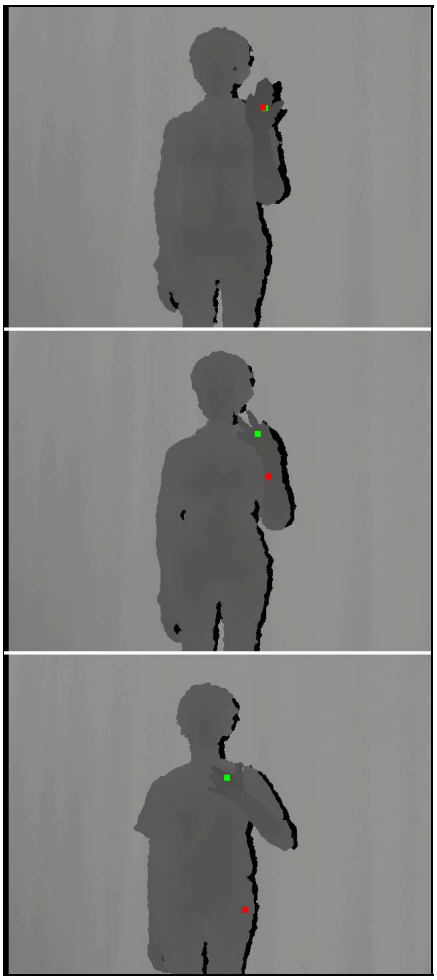


Figure 3.6. Varying accuracy on one-handed signs.

Total processing time for tracking the hand in a single frame, on a PC with a 2.00 GHz Intel(R) Xeon(R) E5406 processor, is about 0.673 seconds. The code is rather unoptimized and implemented in MATLAB R2012a. Processing time can be drastically reduced depending on our I/O and memory management strategy. In the current implementation, we read each frame (saved as a separate file) from the disk, adding a significant computation overhead for just reading and loading each single frame in memory.

3.6 Conclusion and Future Work

We have presented a new hand tracking method that relies depth data and that can accurately identify the location of the gesturing hand in one-handed signs or gestures. The method was compared against one popular skeleton tracking method on a RGB-D dataset of ASL signs. Results show comparable performance while at the same time our method doesn't require an initialization pose like the OpenNI skeleton tracker does. Our experimental evaluation of the readily available skeleton tracker and our depth hand tracking method establish a benchmark for analysis of future detection and tracking algorithms. Another contribution of this chapter is the introduction of an ASL RGB-D video dataset for use in the development and testing of hand detection and tracking methods, as well as in 3D gesture and sign language recognition projects. The dataset provides a large number of gestures that involve one or both hands with varying levels of movement and hand shape complexity and presents an opportunity to develop algorithms that are viable in real world scenarios.

Future work will be to expand the dataset described in section 3.3 until we have multiple examples of all the signs found in [30].

CHAPTER 4

GESTURE RECOGNITION USING DEPTH DATA

4.1 Introduction

This chapter focuses on a specific application, namely a digits gesture recognition system. In human-computer interaction applications, gesture recognition has the potential to provide a natural way of communication between humans and machines. The technology is becoming mature enough to be widely available to the public and real-world computer vision applications start to emerge. However human-computer interaction interfaces need to be as intuitive and natural as possible. The user should ideally interact with machines without the need of cumbersome devices (such as colored markers or gloves [10]) or apparatus like remote controls, mouse and keyboards. Hand gestures can provide an alternative and easy means of communication with machines and could revolutionize the way we use technology in our daily activities. Successful applications of hand gesture systems can be found in various research and industry areas such as: game controlling [34], human-robot interaction [10], virtual environments, smart homes and sign language recognition [35], to name a few. Moreover with the advent and success of Microsoft's new camera, the Kinect^{TM} (see Figure 4.1), it has been clear that computer vision methods and specifically gesture recognition are becoming mature enough to be widely available to the public.

However, in order to create such successful and robust applications there is still much room for improvements. One key challenge for gesture recognition systems is that they must perform in uncontrolled real-world environments. This means heavily cluttered backgrounds with various moving objects and possibly harsh illumination conditions. Most hand gesture recognition systems assume that the gesturing hand can be reliably located in

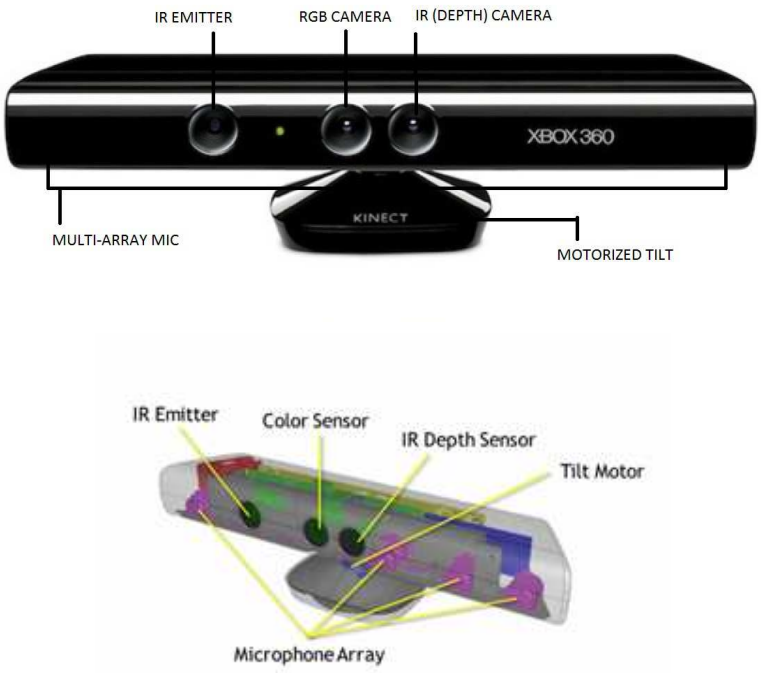


Figure 4.1. Kinect^{TM} camera.

every frame of the input sequence. Unfortunately, in many real life settings perfect hand detection is hard to achieve, especially when we rely on a single source of information such as RGB frames. For example, in Figure 4.2 skin detection yields multiple hand candidates, and the top candidate is often not correct. Skin detection can be affected by varying or harsh illumination. Other visual cues commonly used for hand detection such as motion, edges, and background subtraction [36, 37] would also not perform well in backgrounds with moving objects which could be wrongly classified as moving hands.

In previous Chapter 3 we proposed a method for building a robust hand detector that detects the gesturing hand in a scene by using motion detection based on frame differencing and depth segmentation. Depth segmentation is based on depth intensities for each pixel

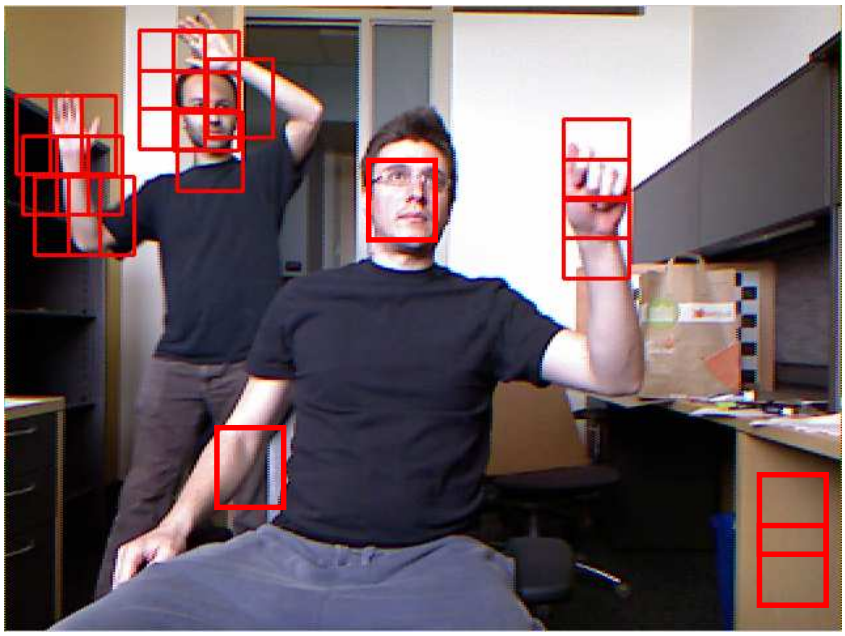


Figure 4.2. Detection of candidate hand regions based on skin color. Clearly, skin color is not sufficient to unambiguously detect the gesturing hand since the face, the non-gesturing hand, and other objects in the scene have similar color. On the other hand, for this particular scene, the gesturing hand is consistently among the top 15 candidates identified by skin detection.

in our images which have been captured with a Kinect^{TM} camera . This hand detector is integrated to the proposed gesture recognition system described in this chapter.

After we compute the location of the hand at each frame of a given video, then we create $2D$ trajectories that represent our gestures. For training purposes, in our database we have precomputed the trajectories for all our gestures that for our experiments correspond to signed digits. Finally we resort to a dynamic programming method, namely DTW [38], in order to compare the test and model trajectories and recognize the gestures. The main advantages of our method are:

- It performs very well even in very challenging environments with the presence of multiple ”distractors” like moving objects, or skin colored objects (e.g., face, non-gesturing hand, background objects).

- It is robust to overlaps between the gesturing hand and the face or the other hand.
- It is translation and scale invariant; the gesture can occur in any part of the image.
- Unlike HMMs and CONDENSATION-based gesture recognition our method requires no knowledge of observation and transition densities, and therefore can be applied even if we have a single example per class.
- Our method can be generalized and applied to recognize a wider range of gestures, other than signs of digits.

The proposed gesture recognition system, to the best of our knowledge, is one of the earliest works that employed depth data from the Kinect™ sensor and provided comparative results using color and depth information. The evaluation of our technique is carried out on a vision-based digit recognition dataset. Each user can sign a digit ranging from 0 to 9 and the goal is to correctly classify the given digit. Similar evaluation frameworks have been followed by other vision-based HCI systems (e.g., the virtual white board by Black and Jepson [39], and the virtual drawing package by Isard [40]), to name a few).

4.2 Related Work

Gesture recognition is a broad research area and a vast amount of literature exists covering all it's different aspects like segmentation, feature extraction and recognition strategies. Mitra and Acharya [41] have carried out a comprehensive survey on gesture recognition covering all the aforementioned aspects. The next two paragraphs highlight some of the most popular feature representation and recognition (or classification) techniques.

Some interesting feature representations, proposed in [42, 43], are velocity histories of tracked keypoints and ballistic dynamics (respectively) which aim to express human gestures and actions. Other popular features used for gesture recognition are 3D Histograms

of Flow (3DHOFs) [44] and Histograms of Gradients (HOGs) [45]. Fanello et al. [46] employed 3DHOFs and Global Histograms of Oriented Gradients (GHOGs) but also added a *sparse coding* feature selection step in order to catch the relevant information inherently underlying the data and in order to discard the redundant information like background or body parts not involved in the gesturing action. *Sparse representation* [47, 46] can been seen as a higher level feature representation. Another very popular higher level feature representation is *bag-of-words* originally proposed by the document classification community. A *bag-of-visual-words* is a sparse vector of the amount of detected local image features of a vocabulary. Csurka et al. [48] have introduced *bag-of-words* to the computer vision community as a novel method for generic visual categorization based on vector quantization of affine invariant descriptors of image patches. Niebles et al. [49] have further exploited *bag-of-words* features to propose a novel unsupervised learning method for action recognition while Dardas and Georganas [50] proposed a real-time system for hand gesture detection and recognition that is trained with SIFT [51] features followed by a vector quantization technique which maps SIFT keypoints from every training image into a unified dimensional histogram vector (bag-of-words) after K-means clustering.

At the recognition level many approaches employ Hidden Markov Models (HMMs) [52, 53, 54] or Dynamic Bayesian Networks (DBNs) [55, 56]. Support Vector Machines (SVMs) is another widely used machine learning technique successfully applied for gesture recognition [46, 57, 50]. Most recently Conditional Random Field (CRF) approaches [58, 59, 60] have also become a standard.

Another popular category of methods for gesture and behavior recognition can be characterized by the use Motion History Images (MHIs). Motion History Images (MHIs) and Motion Energy Images (MEIs) are among the first holistic feature representation methods for behavior recognition [61]. In an MHI H_τ, pixel intensity is a function of the temporal history of motion at that point.

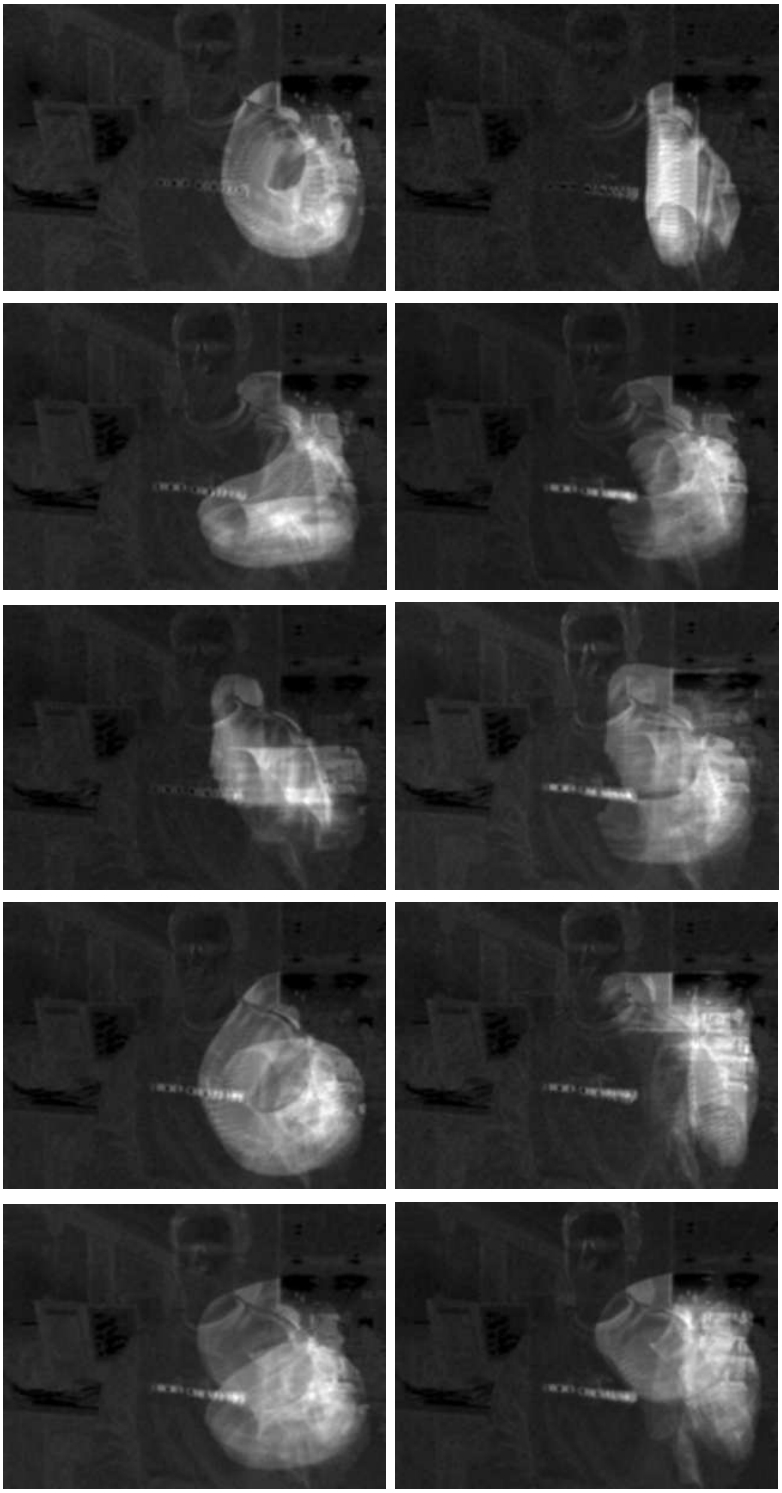


Figure 4.3. Example MHIs representing digits from 0 to 9. MHIs have been computed based on RGB information only.

$$H_{\tau}^{I}(x,y,t) = \begin{cases} \tau, \text{if } |I(x,y,t) - I(x,y,t-1)| > \delta I_{th} \\ max(0, H_{\tau}^{I}(x,y,t-1) - 1), else. \end{cases} \tag{4.1}$$

Here τ is the longest time window we want the system to consider and δI_{th} is the threshold value for generating the mask for the region of motion. The result is a scalar-valued image where more recently moving pixels are brighter. Note that the MEI can be generated by thresholding the MHI above zero. The above mathematical formulation of a MHI and a MEI is described by Davis and Bobick in their seminal paper [61]. They further proposed a statistical description of those images using translation and scale invariant moment-based features (7 Hu moments). For recognition a Mahalanobis distance is calculated between the moment description of the input and each of the known actions. In Figure 4.3 one can see some examples of MHIs for a set of gestures representing digits from 0 to 9.

In [62] Xiang et al. have shown that pixel change history (PCH) images are able to capture relevant duration information with better discrimination performance. The mathematical formulation for a PCH image can be found at the following equation (Eq. 4.2):

$$P_{\varsigma,\tau}(x,y,t) = \begin{cases} \min(P_{\varsigma,\tau}(x,y,t-1) + \frac{255}{\varsigma}, 255) \\ \text{if} D(x,y,t) = 1 \\ \max(P_{\varsigma,\tau}(x,y,t-1) - \frac{255}{\varsigma}, 0) \\ \text{otherwise} \end{cases} \tag{4.2}$$

where $P_{\varsigma,\tau}(x,y,t)$ is the PCH for a pixel at (x,y), $D(x,y,t)$ is the binary image indicating the foreground region, ς is an accumulation factor and τ is a decay factor. By setting appropriate values to ς and τ we are able to capture pixel-level changes over time.

However, using only RGB camera, MHIs can only encode the history of motion induced by the lateral component of the scene motion parallel to the image plane. Ni et al. [63] proposed the use of an additional depth sensor and they develop an extended frame-

work which is capable of encoding the motion history along the depth changing directions. To encode the forward motion history (increase of depth) they introduced the forward-DMHI (fDMHI):

$$H_{\tau}^{fD}(x,y,t) = \begin{cases} \tau, \text{if } D(x,y,t) - D(x,y,t-1) > \delta I_{th} \\ max(0, H_{\tau}^{fD}(x,y,t-1) - 1), else. \end{cases} \tag{4.3}$$

Here, H_t^{fD} denotes the forward motion history image and $D(x,y,t)$ denotes the depth sequence. δth is the threshold value for generating the mask for the region of forward motion. The backward-DMHI (i.e., H_t^{bD}) is generated in a similar way with the thresholding function replaced by $(D(x,y,t) - D(x,y,t-1)) < -\delta th$. In another similar work Kosmopoulos et al. [64] investigated the effects of fusing feature streams extracted from color and depth videos, aiming to monitor the actions of people in an assistive environment. They extracted Pixel Change History Images (PCHs) from RGB streams and backward/forward-DMHI from depth streams. All images were finally represented by 6_{th} order Zernike moments. For classification they reported results on Hidden Markov model classifiers and fusion methods like early, late or state fusion. The use of Kinect™ depthmaps has the main disadvantage of the presence of a significant amount of noise. After frame differencing and thresholding, motion is encoded even in areas where there are only still objects. To tackle this problem one can use a median filtering at the spatial domain. In the temporal domain each pixel value can be replaced by the minimum of its neighbors.

This chapter proposes a trajectory based gesture recognition method employing Kinect™ RGB-D data. Trajectory based methods are typically based on feature vectors regarding hand locations in consecutive frames. Based on this feature representation another wide category of gesture recognition approaches can be defined. Given a video sequence one can define a gesture as the trajectory of the points that correspond to the hand locations

for each video frame. A gesture is therefore a sequence of $2D$ points or a time series. A popular method for comparing time series, that we employ in this chapter, is the Dynamic Time Warping (DTW) [38]. We will describe in more detail the algorithm in section 4.5. DTW has been applied to successfully recognize a small vocabulary of gestures [65, 66].

A main disadvantage of DTW when used in gesture recognition is that it requires a perfect hand detector. For every frame we assume that we have the exact hand location. This assumption of course is hard to be satisfied in uncontrolled real-world environments, as mentioned in the previous section. To address this limitation we need to either build a really robust hand detector, like the one we propose in this chapter, or we need to relax the assumption of a single candidate hand location per frame and allow for multiple detections of candidate hand regions. If we allow for more than one candidates, then we can employ Dynamic Space-Time Warping DSTW [3]. The dynamic space-time algorithm aligns a pair of query and model gestures in time, while at the same time it identifies the best hand location out of the multiple hypotheses available at each query frame.

Another similar approach is multiple hypothesis tracking (e.g., [67]) where multiple hypotheses are associated with multiple observations. The CONDENSATION-based framework can also be applied to gesture recognition [39]. Although in principle CONDENSATION can be used for both tracking and recognition, in [39] CONDENSATION was only used for the recognition part, once the trajectory had been reliably estimated using a color marker. Employing CONDENSATION requires the knowledge of observation and propagation densities for each state of each class model, whereas our proposed method doesn't require such knowledge. Trajectory based methods can be coupled with a HMM framework, as proposed by Sato and Kobyashi [68]. In their method they extend the Viterbi algorithm so that multiple candidate observations can be accommodated at each query frame; the optimal state sequence is constrained to pass through the most likely candidate at every time step. However their approach is not translation invariant and it doesn't

perform well in more challenging setting like the ones we use in our experiments. Next section 4.3 describes our proposed method in more detail.

4.3 Application Overview

Our method is an appearance based and example based gesture recognition method. In our experiments we define 10 classes representing the ten digits from 0 to 9, as shown in Figure 4.4 and 4.5.

Each digit can be formed by a gesture that has been signed by a user and is stored as a video sequence. For each digit class we have several training examples (videos) in our database. More specifically we have 10 different users performing 3 times each gesture digit, thus providing us 300 training examples. To make easier and automate the annotation for the training examples the user wears a colored glove and the background is fairly static and controlled. However we must stress that the same does not apply for the test sequences where the environment is far more challenging.

Given a test video sequence we first need to find the hand location in each frame and create a 2D trajectory. Then we must classify that trajectory as one of our ten classes (Figure 4.6). In this chapter we compare the performance we obtain using a color video, vs. the performance we obtain using a depth video, which provides, for every pixel, the depth. For the color videos, we use a hand detector which combines two visual cues, i.e., color and motion; both requiring only a few operations per pixel. Skin color detection is computationally efficient, since it involves only a histogram lookup per pixel. Similarly, motion detection, which is based on frame differencing, involves a small number of operations per pixel.

For the depth video, we use a hand detector based on motion from frame differencing which we combine with a depth segmentation according to the depth information we have

Figure 4.4. Palm's Graffiti digits .

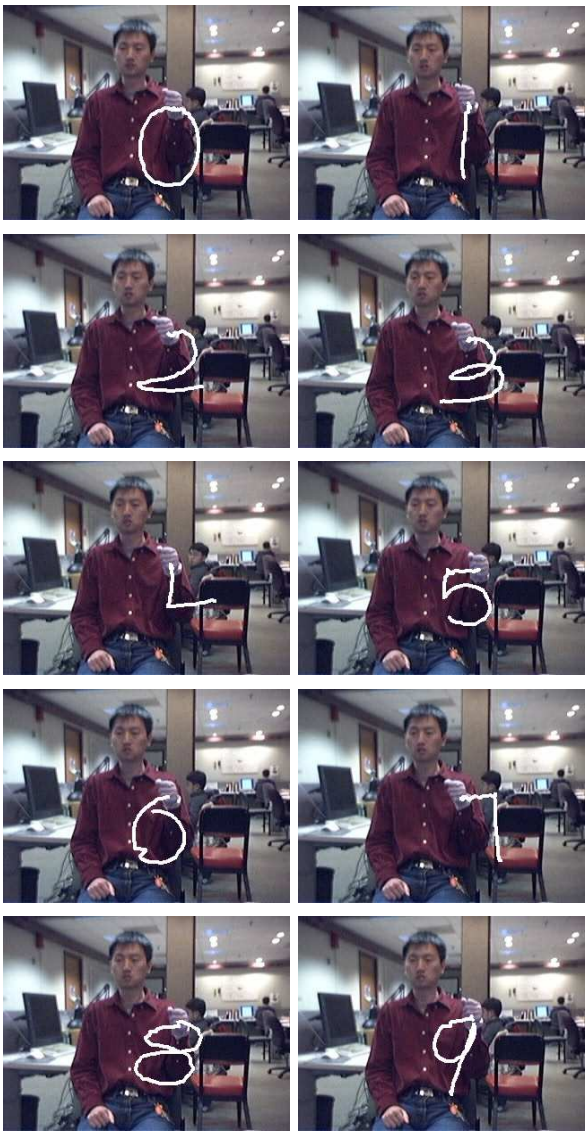


Figure 4.5. Example model digits extracted using a colored glove. We reuse the figure and actual videos from [3].

for each pixel (see previous chapter 3). Our detector can return a single best hand location that will be the input for DTW.

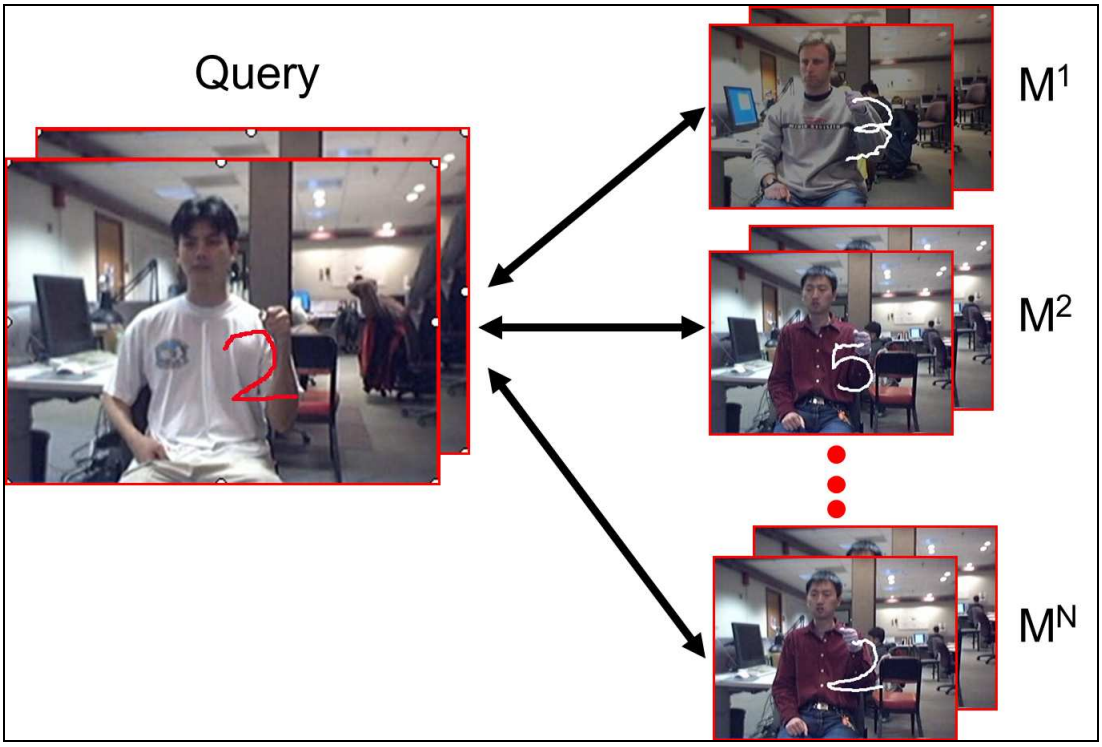


Figure 4.6. Given a test video sequence, we classify it as one of our ten classes by computing the 1NN.

Recognizing the input gesture is done using the nearest neighbor classification framework. The gesture recognition we have just described is depicted in Figure 4.7. The similarity measure that we use for the 1NN scheme (see Figure 4.6) is the score returned by the DTW. The DTW algorithm temporally aligns two sequences, a query sequence and a model sequence, and computes a matching score, which is used for classifying the query sequence. The time complexity of the basic DTW algorithm is quadratic in the sequence length, but more efficient variants have been proposed [69, 38]). In DTW, it is assumed that a feature vector can be reliably extracted from each query frame. In the following sections we describe in detail each module of our method.

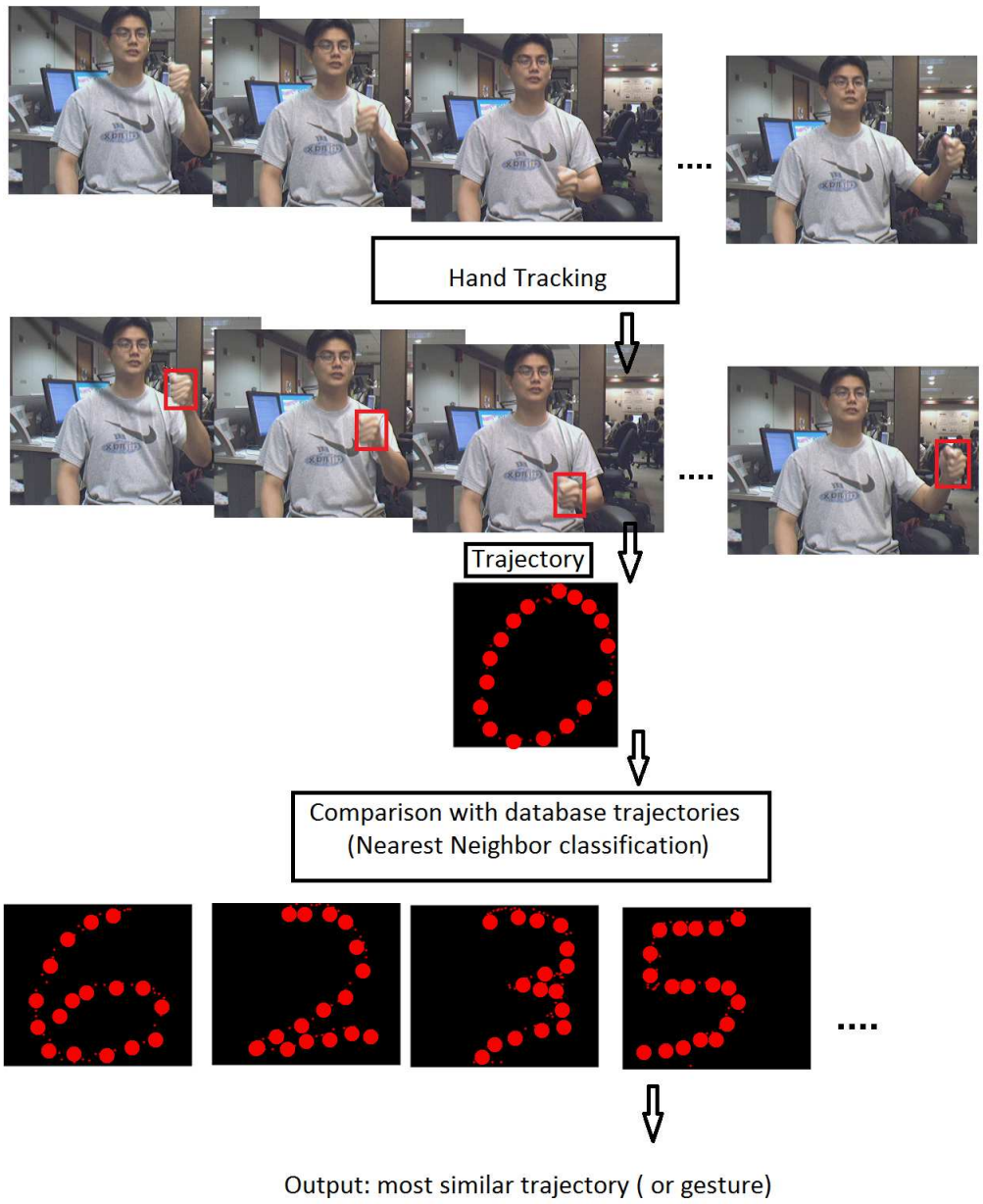


Figure 4.7. A typical bottom-up gesture recognition approach.

4.4 Detection and Normalization

4.4.1 Detection

The first hand detector that we used in our experiments is applied on RGB color images and is based on motion and skin detection. To build the skin color model we use

Figure 4.8. Hand detection in color images: original image (top left), skin detection (top right), frame differencing (middle left), multiplication of skin and frame differencing scores (middle right), top 15 hand candidates (bottom left), single top candidate (bottom right). One can see that the hand detector here fails to detect the hand of the gesturing person.

a generic skin color histogram [70] to compute the skin likelihood image in which each pixel in the frame gets assigned a value denoting the probability of being skin. The motion detector computes a score matrix with the same size as the original image by using frame differencing (frame differencing is the operation of computing, for every pixel, the minimum of two values; the absolute value of the difference in intensity between the current frame and the previous frame and the absolute value of the difference in intensity between the current and the next frame). Then we multiply element by element the motion score matrix with the skin likelihood image to obtain the hand likelihood image. Next we compute for every subwindow of some predetermined size the sum of pixel likelihoods in that subwindow. Then we extract the subwindow with the highest sum. The gesturing hand is typically covered by one or more of these subwindows (See Figure 4.2). Figure 4.8 illustrates examples of input, output, and intermediate steps for this detector.

We use the detector based on RGB information to provide comparative results and a baseline performance for our system. Our proposed gesture recognition however integrates a hand detector based on depth data which has been presented at previous chapter 3. By detecting hands in every frame we create our $2D$ trajectories that are given as input to our recognition algorithm. These trajectories are then normalized in order to achieve translation and scale invariance for our system. The process of normalizing our trajectories is described in more detail at the next subsection 4.4.2.

4.4.2 Normalization

Translation and scale invariance are highly desirable properties for any gesture recognition system. Our method is based on DTW which doesn't inherently satisfy these two properties. Hence we need to pre-process our location features in order to account for differences in translation and scale. When the hand detector is applied, we can segment the hand region. Then we extract our features that will be given as inputs to DTW. We use a ba-

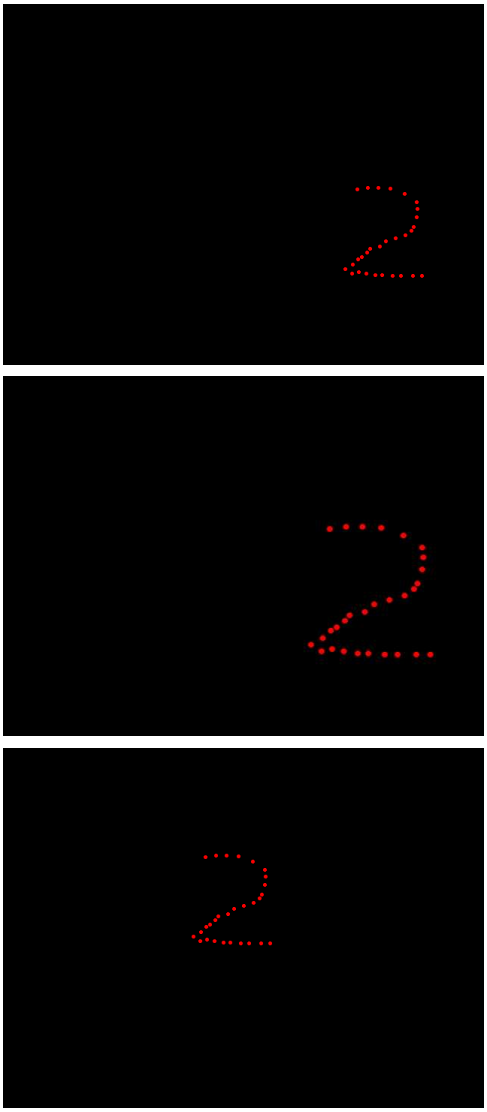


Figure 4.9. On the top image we can see a trajectory representing the digit two. If the user moves towards the camera this may result in a trajectory that appears scaled up (middle image). If the users changes his position with respect to the camera this may result in a trajectory that appears to be translated (bottom image).

sic feature which is the 2D position (x, y) of the segmented region centroid. For any given i_{th} frame a $2D$ feature vector $Q_i = (x, y)$ is extracted. However, the $2D$ feature vector representing position is translation and scale dependent. A user interacting with the gesture recognition system doesn't have to always be located in the same place with respect to the

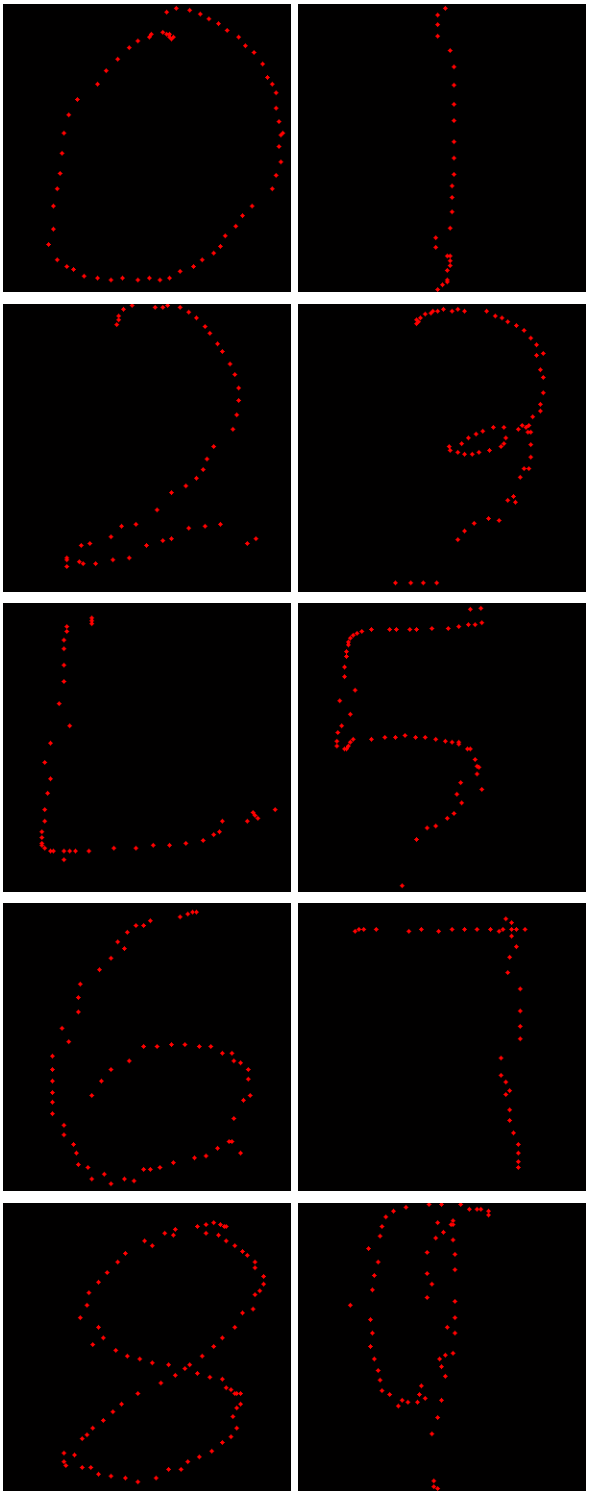


Figure 4.10. In this figure we depict the output of our normalization process. As input to the normalization algorithm we gave the trajectories depicted an Figure 4.11. The actual frame size for the normalized images depicted here is 300×300.

camera. Different placements of the user may result in trajectories with different scale and translation as one can see in Figure 4.9.

In order to render out method scale and translation invariant we must use a normalization step before we give our sequences of 2D vectors as input to DTW.

Normalization is achieved in the following way:

- After we have the set of all 2D points corresponding to hand locations, we can calculate the Minimum Enclosing Circle (MEC) of that set.
- Then we find the bounding square box that has the same center as the MEC and its width length is twice the circle's radius.
- Then we resize the square to a fixed size of 300×300.

By normalizing our features our system becomes translation and scale invariant and increases our recognition rates. The whole normalization pipeline is depicted in Figure 4.12.

In Figure 4.11 we can see some images with the hand trajectories before normalization. Each red pixel represents the actual hand location of the gesturing hand in each video frame. In Figure 4.10 the same trajectories are depicted after the process of normalization.

Another normalization technique that has been proposed by Alon et al. [71] employs a face-centric coordinate system. In order to detect the face of each user at the first frame of the gesture the authors use the face detector of Rowley et al. [72]. The coordinate system is defined so that the center of the face is at the origin, and the diagonal of the face bounding box has length 1. The same scaling factor is applied to both the x and the y direction. However a major disadvantage of this technique is that the user's hand needs to always have the same relative position with respect to the face. This is a rather imposing restriction that impedes the natural interaction between a user and the gesture recognition system.

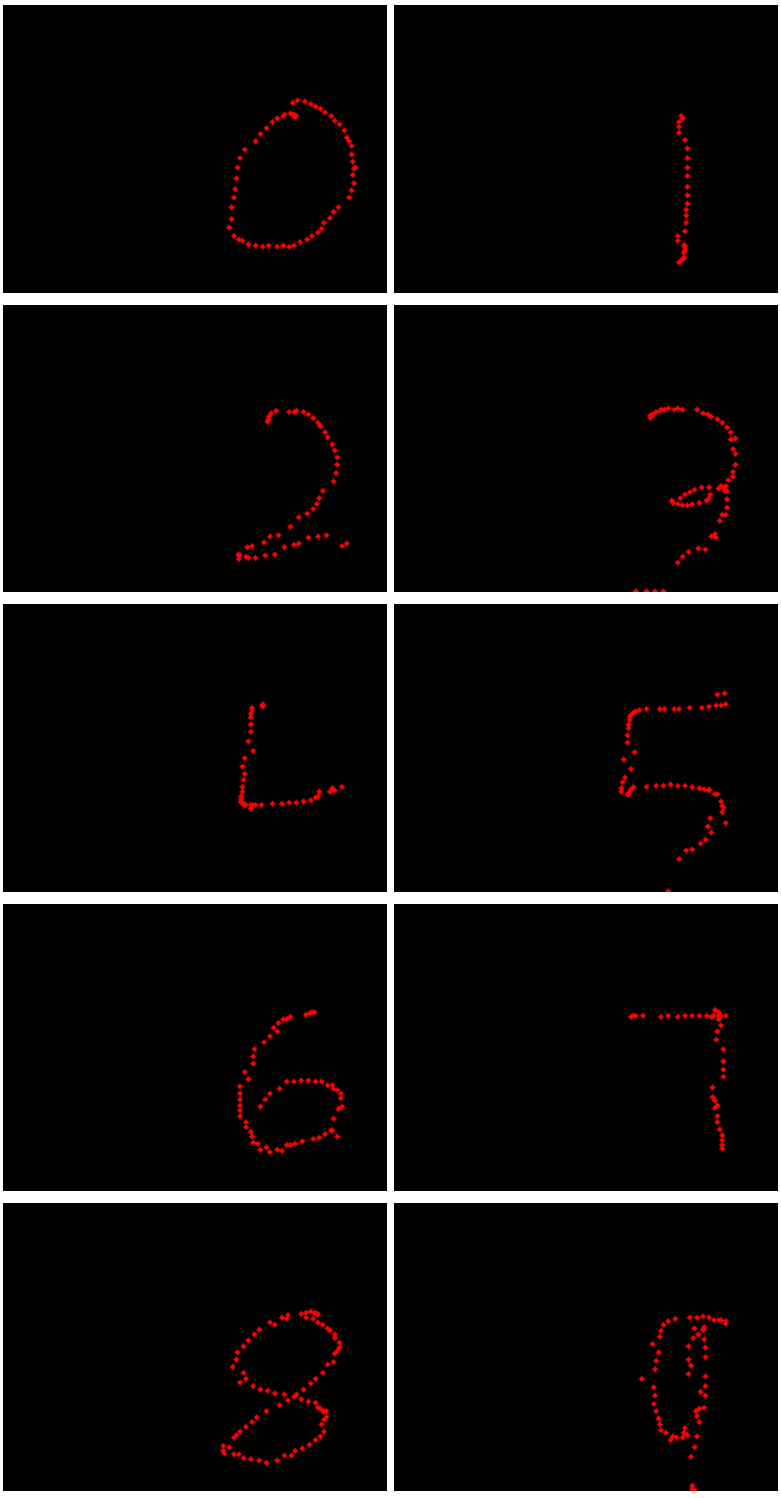


Figure 4.11. The trajectories represent all digits from 0 to 9. Each red pixel represents the actual hand location of the gesturing hand in each video frame. Frames are of size 240×320.

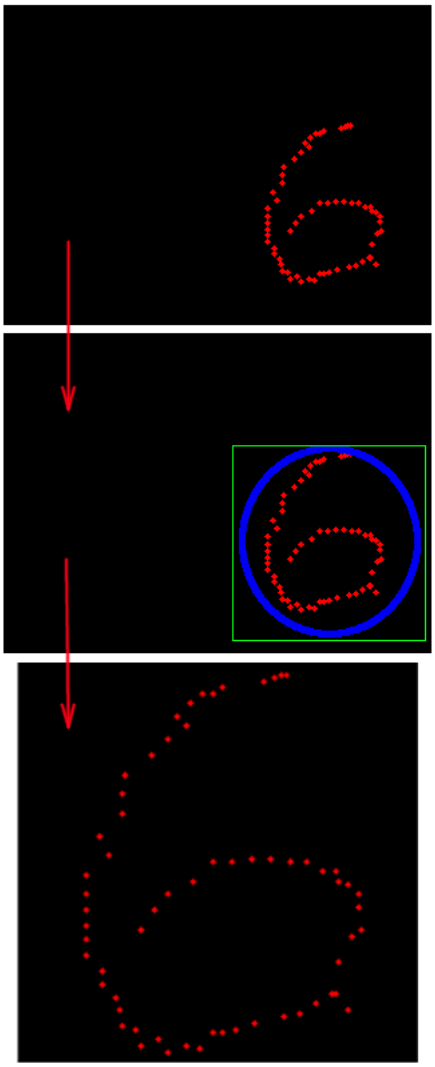


Figure 4.12. This figure depicts our normalization process. On the top image (actual frame size is 240×320) we can see the original 2D trajectory based on the actual hand locations. Then (middle image) we compute the Minimum Enclosing Circle (in red) and the corresponding bounding box (in green). Finally we resize the bounding box to predefined size of 300×300 (bottom image).

The main disadvantage of our normalization technique is that it is sensitive to outliers as depicted in Figure 4.13. However the proposed hand tracking method based on depth data manages to accurately define the hand location in each frame resulting in 2D trajectories without outliers.

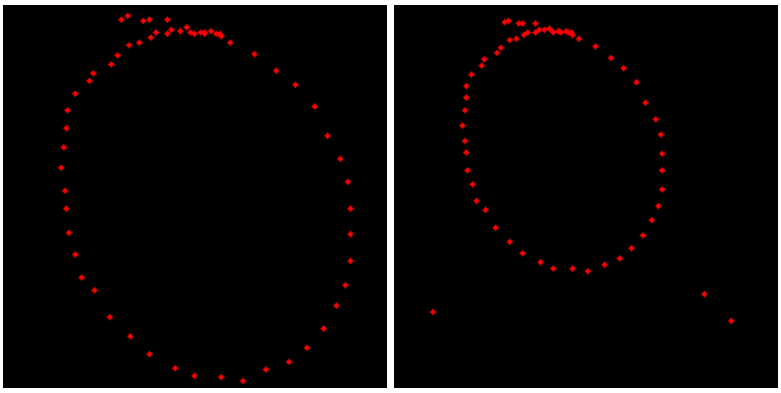


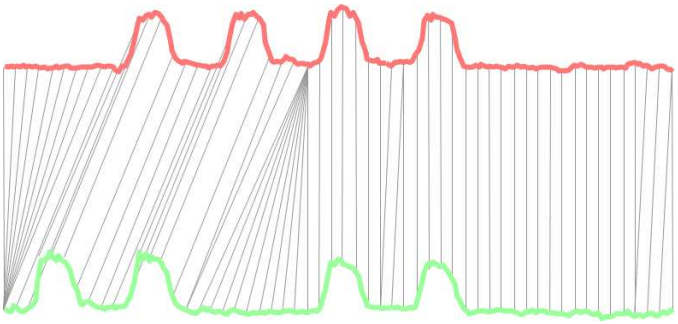
Figure 4.13. To the left image we can see a normalized trajectory. To the right image, is the same trajectory normalized after adding some outliers. Clearly the trajectory has been shifted with respect to the original position and has been scaled down.

4.5 Dynamic Time Warping

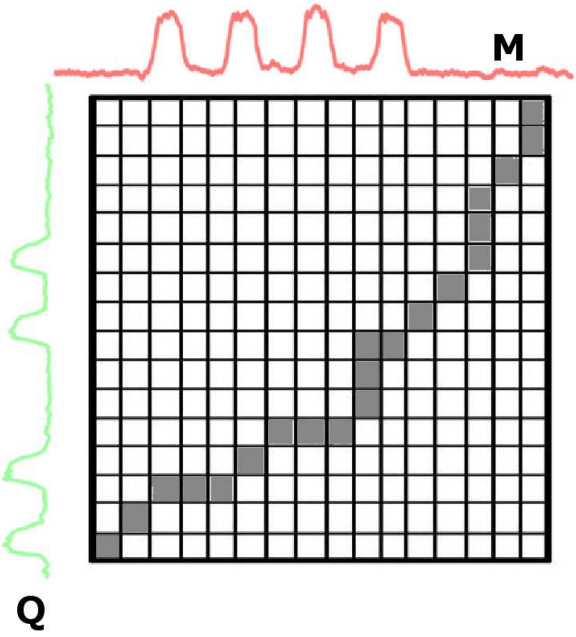
In time series analysis, dynamic time warping (DTW) is a well established algorithm for comparing temporal sequences which may vary in time or speed. DTW addresses the main problem of aligning two sequences in order to compute the most suitable distance measure of their overall difference. Originally employed by the speech recognition community DTW can also be applied to temporal sequences of video data or any type of data that can be expressed as a liner sequence. DTW is a key algorithm for the gesture recognition system described in this chapter. One of the several publications that describe the DTW algorithm is [38]. In this section we briefly describe the algorithm presented in that paper.

Let $M = (M_1, \ldots, M_m)$ be a model video sequence in which each M_i is a feature vector and let $Q = (Q_1, \ldots, Q_n)$ be a query video sequence in which each Q_j is another feature vector. In our experiments each feature vector contains x and y coordinates of the $2D$ hand location. M and Q have a length of m and n, respectively corresponding to the total number of frames of each video sequence. To align two sequences using DTW, we construct an n-by-m matrix where the (i^{th}, j^{th}) element of the matrix contains the distance

(M_i, Q_j) between the two feature vectors M_i and Q_j (i.e., $d(M_i, Q_j) = \|M_i - Q_j\|$. For our experiments d is the Euclidean distance between two feature vectors. Each matrix element (i, j) corresponds to the alignment between the feature vectors M_i and Q_j. This is illustrated in Figure 4.14.



(a) A model M and query Q sequence



(b) Warping matrix defining the alignment for M and Q

Figure 4.14. To align the sequences, we construct a warping matrix and compute the optimal warping path, shown with gray squares.

A warping path W defines an alignment between M and Q. Formally, $W = w_1, \ldots, w_T$, where $\max(m, n) \leq T \leq m + n - 1$. Each $w_t = (i, j)$ specifies that feature vector M_i of the model is matched with query feature vector Q_j. The warping path is typically subject to several constraints:

- Boundary conditions: $w_1 = (1, 1)$ and $w_T = (m, n)$. This requires the warping path to start by matching the first frame of the model with the first frame of the query, and end by matching the last frame of the model with the last frame of the query.
- Temporal continuity: Given $w_t = (a, b)$ then $w_{t-1} = (a', b')$, where $a - a' \leq 1$ and $b - b' \leq 1$. This restricts the allowable steps in the warping path to adjacent cells along the two temporal dimensions.
- Temporal monotonicity: Given $w_t = (a, b)$ then $w_{t-1} = (a', b')$ where $a - a' \geq 0$ and $b - b' \geq 0$. This forces the warping path sequence to increase monotonically in the two temporal dimensions.

An exponential number of warping paths can be found that adhere to the above restrictions. However DTW computes the optimal path that will minimize the following warping cost:

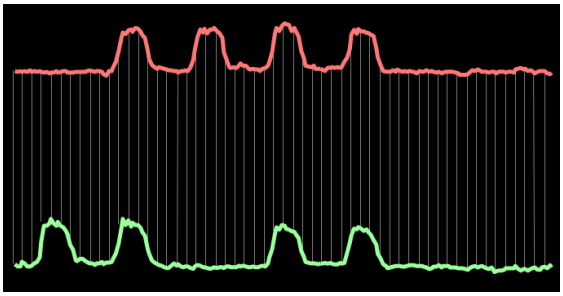
$$DTW(M, Q) = min\{\sqrt[2]{\sum_{t=1}^{T} w_k}\} \tag{4.4}$$

This path can be found using dynamic programming (see Algorithm 2) which computes the cumulative distance $\gamma(i, j)$ as the distance $d(i, j)$ found in the current cell and the minimum of the cumulative distances of the adjacent elements:

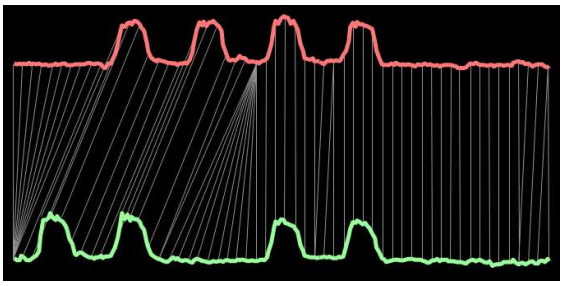
$$\gamma(i, j) = d(M_i, Q_j) + min\{\gamma(i - 1, j - 1), \gamma(i - 1, j), \gamma(i, j - 1)\} \tag{4.5}$$

The time and space complexity of DTW is $O(nm)$.

The Euclidean distance between two sequences can be seen as a special case of DTW where the kth element of W is constrained such that $w_k = (i, j)_k, i = j = k$. Note that it is



(a) Euclidean distance



(b) DTW

Figure 4.15. Note that the two sequences are quite similar but not aligned in the time axis. Euclidean distance doesn't take into account the fact that the two time series are out of phase. Hence it will produce a pessimistic dissimilarity measure. DTW alignment will provide a more intuitive distance measure.

only defined in the special case where the two sequences have the same length (see Figure 4.15).

Given warping path element $w_t = (i, j)$, we define the set $N(i, j)$ to be the set of all possible values of w_{t-1} that satisfy the warping path constraints (in particular continuity and monotonicity):

$$N(i, j) = \{(i - 1, j), (i, j - 1), (i - 1, j - 1)\} \tag{4.6}$$

We assume that we have a cost measure $d(i, j) \equiv d(M_i, Q_j)$ between two feature vectors M_i and Q_j. In our experiments $d(i, j)$ is the Euclidean distance. DTW finds the optimal path W^* and the global matching score D^* as described in Algorithm 2.

input : A sequence of model feature vectors $M_i, 1 \leq i \leq m$, and a sequence of query feature vectors $Q_j, 1 \leq j \leq n$.

output : A global matching cost D^*, and an optimal warping path $W^* = (w_1^*, \ldots, w_T^*)$.

$D(1,1) = d(1,1)$ // Initialization

$j = 1$ **for** $i = 2 : m$ **do**

  $D(i,1) = D(i-1,1) + d(i,1)$

end

$i = 1$ **for** $j = 2 : n$ **do**

  $D(1,j) = D(1,j-1) + d(1,j)$

end

for $i = 2 : m$ **do**

  // Iteration **for** $j = 2 : n$ **do**

    $w = (i,j)$

    for $w' \in N(w)$ **do**

      $C(w',w) = D(w')$,

    end

    $D(w) = d(w) + \min_{w' \in N(w)} C(w',w)$

    $b(w) = \text{argmin}_{w' \in N(w)} C(w',w)$

  end

end

$k^* = \text{argmin}_k \{D(m,n)\}$ // Termination

$D^* = D(m,n,)$

$w_T^* = (m,n,)$

$w_{t-1}^* = b(w_t^*)$ // Backtrack

Algorithm 2: The DTW algorithm

4.6 Experiments and Results

To evaluate the performance of our method we have created a hand-signed digit recognition system. The training videos that we use are publicly available, as described in [71]. In that data, the trajectories can be easily found, since the persons signing the digits are wearing a green colored glove. This is a convention that we allow only on our training data which we then preprocess offline in order to create fine trajectories. However this is not the case for the testing data. Our test video sequences have been captured in some really challenging settings so as to measure the robustness of our proposed and previous methods. The test data have been collected with the Kinect^{TM} camera using an image resolution of 640×480. In more detail our datasets have been organized as follows:

- Training examples: 300 digit exemplars (30 per class) were stored in the database (See Figure 4.5). A total number of 10 users have been employed to collect all training data
- Test gestures: 40 digit exemplars (20 per class) were used as queries. For 20 of them the users were wearing short-sleeve shirts contrary to most gesture recognition methods. We will refer to this half of the test set as the *easy test set*. For the remaining 20 exemplars, we have created even more challenging conditions, with people and various objects moving constantly in the background. In this way we want to demonstrate that most of the previous methods fail while our proposed method remains robust even in the harshest conditions. We will refer to this half of the test set as the *hard test set*. A total number of 2 users have been employed to collect all test data.

It is important to note that recognition was performed in a *user-independent* manner: we never use a model video sequence as a test query. The test videos have been collected from users that do not appear in the training videos. Using a hand detector we extract a trajectory from a test query. Then we compare that trajectory with all the pre-computed

training trajectories and using 1-NN Nearest Neighbor Classification we classify our test signed digit as one of the 10 classes, ranging from 0 to 9.

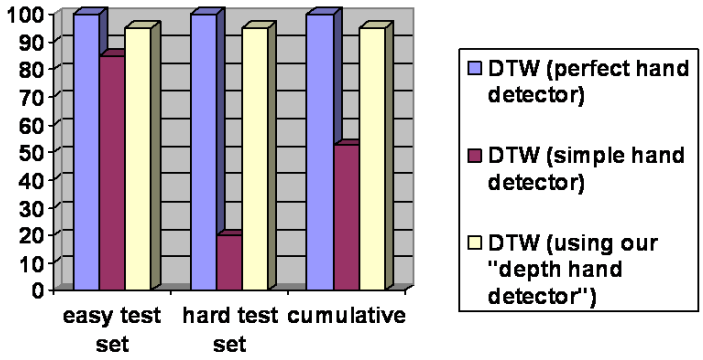


Figure 4.16. Results.

First we test the performance of DTW given that we have a perfect hand detector for our test sequences. To extract the trajectories in this case we have resorted to manual annotation. Naturally, the gesture recognition is 100% accurate and all digits are classified correctly. Then we test DTW by employing another detector which uses motion and skin color to locate the hand in each frame. This will also serve as a baseline performance for our system. For the *easy test set* we achieve an accuracy of 85% while for the *hard test set* the accuracy reduces drastically down to 20% emphasizing the weakness of previous methods in such challenging environments. Finally, our method achieves an accuracy of 95% for both test sets. All results are depicted in Figure 4.16.

Total processing time for tracking the hand in a single frame, on a PC with a 2.00 GHz Intel(R) Xeon(R) E5406 processor, is about 0.673 seconds. The code is rather unoptimized and implemented in MATLAB R2012a. Processing time can be drastically reduced depending on our I/O and memory management strategy. In the current implementation, we read each frame (saved as a separate file) from the disk, adding a significant computation

overhead for just reading and loading each single frame in memory. The computation time for the recognition algorithm, including the normalization step, is about 19 seconds for a random gesture containing 68 frames. The normalization step takes as input the whole set of $2D$ gesturing points. This introduces a small latency to our recognition algorithm which requires that the whole gesture has to be first captured.

4.6.1 Conclusion and Future Work

This thesis chapter has introduced a translation and scale invariant gesture recognition method that can achieve high performance even in challenging environments with backgrounds full of moving people and objects. The contribution of this chapter is a gesture recognition system that integrates a hand detector based on motion detection and depth segmentation that can accurately locate the hand at each frame. On top of the detector the features are normalized and then the Dynamic Time Warping algorithm is employed in order to recognize the gestures. Incorporating translation and scale invariance makes the proposed system more user friendly since the user has fewer restrictions as to where exactly he needs to be placed with respect to the camera. However some restrictions remain and one of them is viewpoint invariance. The current method assumes that the user always faces the camera so as to capture a frontal view of the gesture. An extension of this approach that can handle arbitrary camera viewpoints is presented in Chapter 5. Another open problem that needs investigation is temporal segmentation. In the current approach the beginning and end frame for each gesture is manually annotated. Automating this procedure (e.g., by using a distinct pose for the non-gesturing hand) remains future work. Finally, we would like to expand our gesture recognition system in order to accommodate more challenging gestures from other domains such as the American Sign Language. All the aforementioned future directions address interesting research problems but implementing them is out of the scope of the current thesis.

CHAPTER 5

VIEWPOINT INVARIANT GESTURE RECOGNITION USING RGB-D

5.1 Introduction

Towards developing NUI systems the ultimate goal is to offer to the user a type of interaction that is as natural and unconstrained as possible. Hence, two highly desirable properties for any action/gesture recognition systems is scale and translation invariance. The method we proposed at previous chapter 4 satisfies both properties. However another essential property that needs to be addressed is viewpoint invariance. Typically, most gesture recognition approaches assume that the gesturing plane is fronto-parallel to the camera image plane. However this imposes a restriction to the user as he can not be placed freely under an arbitrary viewpoint with respect to the camera. Even though there is a vast amount of literature on gesture and action recognition, few approaches explicitly target the viewpoint invariance property. At the same time traditional methods rely on $2D$ and most recently on $2.5D$ data where there might be loss of information from the original $3D$ gestures. Relying on $2D$ information means that the analyzed trajectory is just a projection onto the image plane of the actual $3D$ gesture. Depending on the viewpoint and after the projection we might loose useful information that was previously encoded in the $3D$ data.

In this chapter we will extend our method to exploit $3D$ point clouds and handle cases where the user is not standing fronto-parallel to the camera. Instead we allow the user to perform gestures in such a way that the gesturing plane might be rotated with respect to the camera image plane. We consider rotations along the camera coordinate system y axis but our method can also handle differences along the x axis. To estimate these angles we use the unit length normal vector of the gesturing plane. In our experiments we recorded videos

and we managed to recognize gestures from a set of different viewpoints: $\{\pm 45^{\circ}, \pm 75^{\circ}\}$. Our system has some attractive properties which can be summarized as follows:

1. It is trained from videos captured under one specific camera viewpoint but it can be tested with gestures captured under arbitrary camera viewpoints. In our experiments we opt to train our system with a camera viewpoint where the user is standing fronto-parallel to the image plane. For testing the videos are captured under the following set of viewpoints $\{\pm 45^{\circ}, \pm 75^{\circ}\}$.
2. It is all-together translation, scale and viewpoint invariant. To the best of our knowledge few gesture recognition methods satisfy all these three properties at the same time
3. It employs an affordable, commercially available sensor (i.e., *Microsoft Kinect*^{TM}) as opposed to an expensive laboratory sensor or a cumbersome calibrated multi-camera set-up

In the next section 5.2 we will present related work for view-invariant action/gesture recognition.

5.2 Related Work

One of the earliest works on viewpoint invariant gesture recognition has been proposed by Weinland et al. [73]. They introduced a free-viewpoint representation of *Motion History Volumes* (MHV) for human actions using an experimental setup of multiple-calibrated and background-subtracted video cameras. Ultimately their goal was to build free-viewpoint class models from view-invariant motion descriptors. To acquire the motion descriptors the main assumption is that viewpoint variations are expressed with respect to the vertical axis of the gesturing body. Then the motion templates (MHV) can be expressed invariantly (translation and rotation) around the body axis by using Fourier-magnitudes and

cylindrical coordinates. After computing motion descriptors in a view-invariant way the final task is to recognize or classify the actions. This is typically done by learning statistical models of the temporal sequencing of the descriptors. Classification is achieved using dimensionality reduction (PCA) coupled with Mahalanobis distance and linear discriminant analysis (LDA). Experimental results are reported on a set of 11 actions (IXMAS dataset) captured in a laboratory environment with five Firewire cameras that were calibrated and synchronized.

In [74] Souvenir and Babbs use the same dataset (i.e IXMAS) but for training purposes they animate the visual hull of gesturing body and project the silhouette onto 64 evenly spaced virtual cameras located around the body's vertical axis. They model appearance of actions as a function of the camera viewpoint. More specifically they formulate actions by learning a low-dimensional representation of high-dimensional $\mathcal{R}$ transform surfaces, which lie on or near a low-dimensional manifold. In other words, $\mathcal{R}$ transform surfaces vary as a function of the viewpoint parameter which is learned with manifold learning.

All the above methods rely on a multi-camera experimental setup where all cameras need to be calibrated and synchronized. Such an elaborate laboratory setting can not be applied when we need to observe actions in scenarios within unconstrained real environments (e.g., smart homes or sign language recognition applications). At the same time the training phase for many different viewpoints can be an extremely strenuous process.

More recently, Holte et al. [75] have proposed a method that doesn't require a set of multiple calibrated cameras. Instead they employ a Time-of-Flight (ToF) range camera, namely the SwissRanger SR4000. This range camera can provide RGB-D synchronized image frames similarly to the Kinect^{TM} device, as described in previous chapters. The authors represent gestures as an ordered sequence of $3D$ *motion primitives*. Since the focus of that work is on hand gestures a segmentation based on motion is applied in order to isolate

arms from the rest of the body. More specifically the arms are extracted by using a $3D$ version of optical flow in order to compute velocity annotated point clouds that are finally represented by their *motion context*. Motion context is an extended version of regular shape context. The $3D$ motion primitives can be expressed in an invariant way with respect to rotation around the vertical axis. Towards this end the authors choose to use spherical harmonic basis functions, yielding a harmonic motion context representation. In each video frame, the observed data can be represented by a primitive (if any). After identifying primitives from consecutive frames a discrete recognition problem can be constructed, since a video sequence of range data will be converted into a string containing a sequence of symbols, each representing a primitive. After pruning the string a probabilistic Edit Distance classifier is applied in order to determine which gesture best represents the pruned string.

Our proposed method for view invariant gesture recognition is trained with videos from one camera viewpoint and tested with videos taken from completely different viewpoints ranging from -75° to $+75^{\circ}$. The main advantages of our approach over [75] are the following:

1. We use an affordable, commercially available sensor (i.e., Microsoft Kinect$^{\text{TM}}$) as opposed to an expensive laboratory sensor like the SwissRanger SR4000.
2. We can recognize gestures captured from a wider set of viewpoints, ranging from -75° to $+75^{\circ}$ as opposed to a smaller set of viewpoints (see [75]) ranging from -45° to $+45^{\circ}$.

A more comprehensive survey regarding viewpoint invariant gesture/action recognition methods can be found at [76]. The remainder of this chapter is organized as follows. In the next section 5.3 we describe in more detail our method and in section 5.4 we present the experimental setup and results. Finally at section 5.5 we conclude and discuss possible future work.

5.3 Methodology

We propose a viewpoint invariant gesture recognition system that can be seen as an extension of our system presented at previous chapter 4. The main advantage here is that we can recognize gestures extracted from videos that have been captured under varying viewpoint directions of the camera with respect to the user. For each video we employ RGB and depth information in order to determine the $2D$ hand locations of the gesturing hand. Relying on $2D$ information means that the analyzed trajectory is just a projection onto the image plane of the actual $3D$ gesture. Depending on the viewpoint and after the projection we might loose useful information that was previously encoded in the $3D$ data. To address this issue we will define our trajectories in both $2D$ and $3D$ space. By combining the registered and synchronized RGB and depth frames it is possible to construct $3D$ point clouds and thus estimate $3D$ hand locations. The framework for capturing and processing $3D$ point clouds is offered by the Point Cloud Library (PCL) [77], which is a large scale, open project for $2D/3D$ image and point cloud processing. By using the RGB and depth frame along with the camera's (i.e., Kinect™) intrinsic and extrinsic parameters PCL offers a fully $3D$ reconstructed point cloud. An interesting property of the new point cloud is that for each i, j pixel of the original VGA resolution image we can now have access to the corresponding $3D$ coordinates x, y and z expressed in meters. So for any given pixel p_i we know all-together:

- $2D$ spatial information, i.e., row and column (i, j respectively)
- Color information (i.e., RGB value for p_i)
- $3D$ spatial information based on x, y, z coordinates, expressed in a $3D$ coordinate system where the origin coincides with the optical center of the Kinect™ RGB sensor. In Figure 5.1 we can see a snapshot of a rendered point cloud along with the $3D$ coordinate system. Unit of measurement is in meters. Red axis is x, green axis is y

and finally blue axis z. PCL defines the $3D$ coordinate system following the same conventions as the *pinhole camera model*

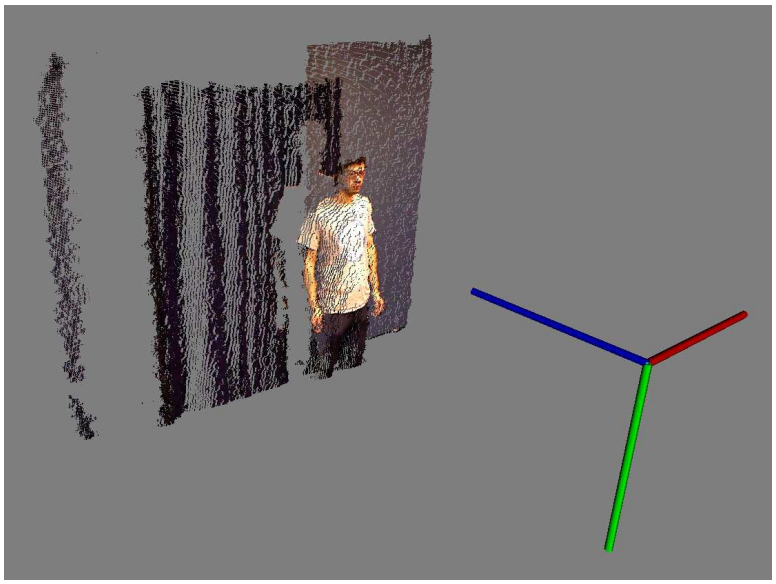


Figure 5.1. This figure depicts a point cloud with a user performing a gesture. The origin of the $3D$ coordinate system coincides with the optical center of the Kinect^{TM} RGB sensor. Unit of measurement is in meters. Red axis is x, green axis is y and finally blue axis is z. PCL defines the $3D$ coordinate system following the same conventions as the *pinhole camera model*.

By using PCL, any extracted $2D$ trajectory can be also expressed as a $3D$ trajectory now. Towards quantifying the differences between various camera viewpoints we first need to define the *gesturing plane*. This can be computed by fitting a plane for the points that belong to the $3D$ trajectory and correspond to the actual hand locations in each frame. This imposes the restriction that all gesturing points need to be co-planar. Future work will be to define the *gesturing plane* by fitting a plane for the points that belong to the torso of the user. In this way our method could handle a wider set of gestures without requiring them to be co-planar. We express $3D$ planes in their *Hessian Normal form* (see Equation 5.1),

$$\hat{n} \cdot x = -p \tag{5.1}$$

where p is the distance of the plane from the origin and vector $\hat{n} = (n_z, n_y, n_z)$ is the unit normal vector (normal to the surface of the plane) and it's components are defined as shown respectively at equations 5.3, 5.4, 5.5. The general equation of a $3D$ plane is defined at Eq. 5.2.

$$ax + by + cz + d = 0 \tag{5.2}$$

$$n_x = \frac{a}{\sqrt{a^2 + b^2 + c^2}} \tag{5.3}$$

$$n_y = \frac{b}{\sqrt{a^2 + b^2 + c^2}} \tag{5.4}$$

$$n_z = \frac{c}{\sqrt{a^2 + b^2 + c^2}} \tag{5.5}$$

$$p = \frac{d}{\sqrt{a^2 + b^2 + c^2}} \tag{5.6}$$

By using the unit length normal vector $\hat{n}$ of the *gesturing plane* we can compute the camera viewpoint angle θ with respect to the user. Let's denote with $\hat{n}'$ the projection of $\hat{n}$ onto the XZ plane. We define as θ the rotation angle between $\hat{n}'$ and the unit length normal vector $\hat{z}$ that lies on z axis. Rotation angle θ can quantify differences between various camera viewpoints. But, most importantly we can use θ to apply a $3D$ transformation for the points comprising the $3D$ trajectory. Towards that end we construct the 3×3 rotation matrix $R_y(\theta)$ (see Eq. 5.7) as follows:

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{5.7}$$

After multiplying to the left the transformation matrix $R_y(\theta)$ with all $3D$ trajectory points the angle θ between image plane and gesturing plane will get reduced to zero ($\theta = 0^\circ$). Matrix $R_y(\theta)$ accounts for rotations with respect to the camera y-axis but in a similar fashion we can also handle cases where we also have rotations with respect to the camera x-axis. After the transformation we can regard the $3D$ gesture being captured as if the user was standing fronto-parallel to the camera. We can extract the corresponding $2D$ gesture by projecting all points onto the XY plane. We illustrate the effect of that transformation in Figures 5.3 and 5.4. First, in Figure 5.3 be show the $2D$ trajectories that have been generated by detecting hand locations in a video where $\theta = 75^\circ$ (see also Figure 5.2). It is evident that useful information from the original $3D$ gestures has been lost.

Figure 5.2. This figure depicts a user performing a gesture from a camera viewpoint such as that $\theta = 75^\circ$.

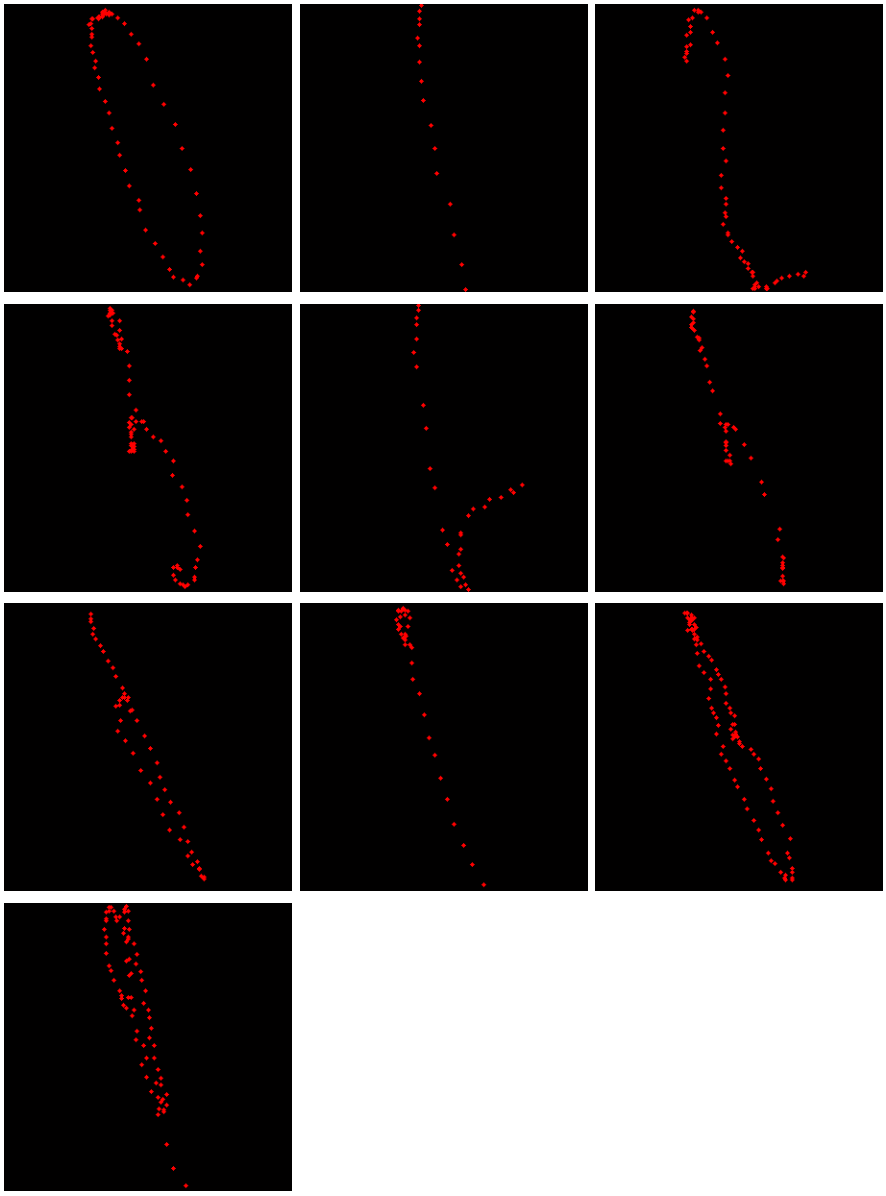


Figure 5.3. In this figure we depict $2D$ trajectories that have been created by detecting hand locations in $2D$ RGB-D images. The original $3D$ gestures represent hand-signed digits from 0 to 9. The user is not facing the camera from a frontal view but from a viewpoint with $\theta = 75^{\circ}$. The actual frame size for the depicted images is 300×300.

In Figure 5.4 we demonstrate the effect of the $3D$ transformation. The original $3D$ gestures here are the same as in previous figure 5.3. We apply the transformation so as the *gesturing plane* becomes parallel to the image plane. Finally we depict the $2D$ trajectories created by projecting the $3D$ points onto XY plane.

Since our system is trained with fronto-parallel $2D$ trajectories we can classify the transformed $2D$ gestures with Nearest Neighbor classification employing DTW as a similarity measure. The general framework for the proposed view invariant gesture recognition method can be summarized as follows with the following steps:

1. In each frame we compute the $2D$ hand locations, manually or automatically with a hand tracker. In the end we have a $2D$ trajectory representing our gesture.
2. By using PCL we can express the $2D$ trajectory in $3D$.
3. Fit a plane for the $3D$ trajectory by using RANSAC. Gesturing plane is expressed in *Hessian Normal form*.
4. Find the transformation matrix that aligns the unit length normal vector of the gesturing plane with the z unit length vector of the image coordinate system.
5. Apply that transformation to all $3D$ points of the trajectory and then project to the XY image plane.
6. Normalize the $2D$ trajectory (see subsection 4.4.2 of chapter 4)
7. Classify the new transformed and normalized $2D$ trajectory, by giving it as input to the DTW algorithm and find the best match from our training database. The Nearest Neighbor classification scheme we employ here is the same as in the previous Chapter 4.

The next subsection describes RANSAC algorithm which is used for computing the parameters of the *gesturing plane*.

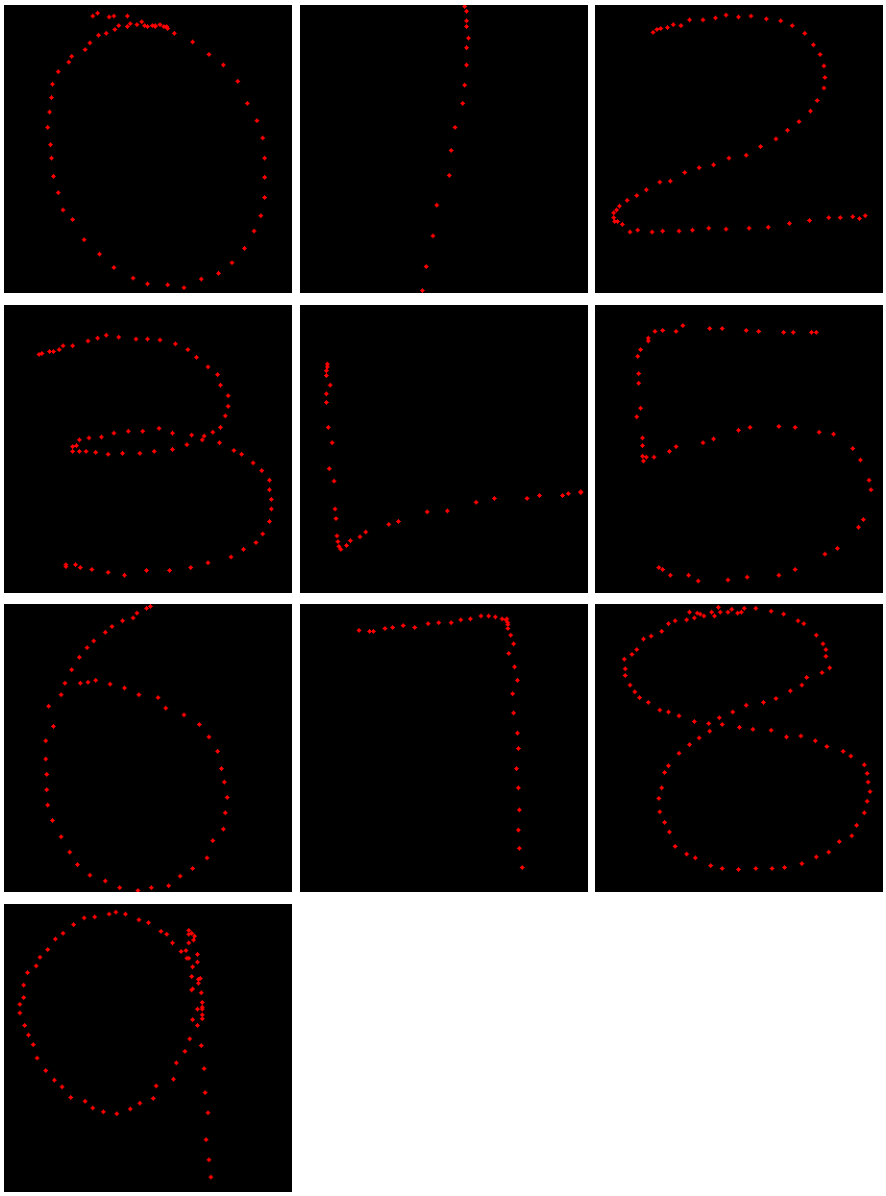


Figure 5.4. In this figure we demonstrate the effect of the $3D$ transformation. The original hand-signed digit $3D$ gestures are the same as in previous figure 5.3. We apply the transformation so as the *gesturing plane* becomes parallel to the image plane. Finally we depict the $2D$ trajectories created by projecting the $3D$ points onto XY plane. The actual frame size for the depicted images is 300×300.

5.3.1 RANSAC

The RANdom SAmple Consensus (RANSAC) algorithm proposed by Fischler and Bolles [78] is an iterative method for fitting of parametric models which is specifically designed to be robust in the presence of many data outliers. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed. RANSAC is a re-sampling technique that generates model hypotheses by employing the minimum number of input data (data points) required to estimate the underlying model parameters. So for example, when estimating $3D$ planes (as in our experiments) it requires only three data points to estimate the plane parameters. This is in contrast with traditional sampling techniques that first require a large set of input data for the initial parameters estimation and then prune out outliers. RANSAC first uses the smallest possible set of input data and then expands it in a iterative fashion with more data that are consistent with the initial model parameters. The basic algorithm is formally described here:

Lets denote with p the probability that at least one set or random data items contains only inliers. Typically p is set to 0.99. Lets denote with p_{inlier} the probability that any selected data item is an inlier. Lets denote with $p_{outlier}$ the probability that a selected data item is an outlier. We know that:

$$p_{outlier} = 1 - p_{inlier} \tag{5.8}$$

Remember that N is the number of minimum required data items in order to define the model parameters. The number of iterations L can be determined based on the following equation:

$$1 - p = (1 - (p_{inlier})^N)^L \tag{5.9}$$

input : A fitting problem with parameters $\vec{x}$, as set of total M input points, the number N of minimum required points to define the model parameters, L is the maximum number of iterations, τ and κ are predefined thresholds

output : The estimated model parameters and the set of inlier points

for $i = 1 : L$ **do**

  select N minimum required data points randomly;

  estimate $\vec{x}$ (i.e., model parameters);

  find number D of all data items (out of M) that are consistent with our model parameters, given a tolerance κ;

  if $\frac{D}{M} > \tau$ **then**

    re-estimate the model parameters $\vec{x}$ using all D data items;

    exit and return $\vec{x}$ along with the D data items;

  end

end

fail if you get here;

Algorithm 3: The RANSAC algorithm

$$L = \frac{log(1-p)}{log(1-(1-p_{outlier})^N)} \tag{5.10}$$

One of the main advantages of RANSAC is that it can robustly estimate the model parameters even in the presence of a large amount of outliers in the original data-set. On the other hand because of the non-determinist nature of the algorithm there is no upper bound for the computation time for estimating the correct model parameters. The more iterations performed the higher the probability of an accurate model being estimated. At the same RANSAC requires the setting of problem-specific thresholds. Contrary to many of the common robust estimation techniques, such as M-estimators and least-median squares that have been derived from the statistics literature, RANSAC was developed from within

the computer vision community. In our experiments we employ RANSAC to compute $3D$ gesturing planes. Next section 5.4 presents experimental results assessing the performance of the proposed method.

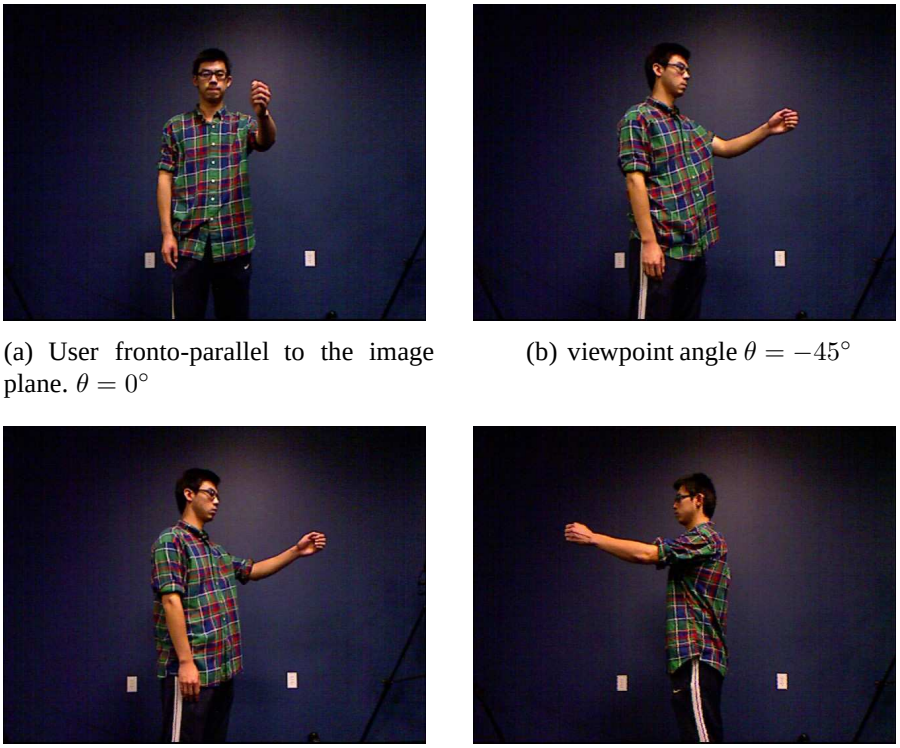
5.4 Experimental Results

One of the main advantages of our viewpoint invariant gesture recognition method is that it can be trained from one camera viewpoint and tested under various different camera viewpoints. In our experiments we choose to train our system by using a camera viewpoint where the user is standing fronto-parallel to the image plane. The training videos that we use are the same as in previous Chapter 4. All gestures represent hand-signed digits from 0 to 9. Our training database consists of 300 digit exemplars (30 per class) expressed as normalized $2D$ trajectories. A total number of 10 users have been employed to collect all training data.

5.4.1 Testing Dataset

For testing purposes we have captured videos under various camera viewpoints ranging from -75° to $+75^{\circ}$. More specifically we have tested our system under the following 4 different viewpoint angles $\{\pm 45^{\circ}, \pm 75^{\circ}\}$. Figure 5.5 shows a user performing gestures under some of the aforementioned viewpoint angles.

At total of 3 users participated in the creation of our test dataset. Each user has performed gestures with 4 different camera viewpoints. For each viewpoint and user we have collected 10 gestures representing hand-signed digits (from 0 to 9). We have collected a total of $3 \times 4 \times 10 = 120$ testing gestures and a total of 6286 frames. Since the focus of this Chapter is on the viewpoint invariant recognition algorithm we have manually identified the original $2D$ hand locations in the test video sequences. Implementing an automatic hand tracker that can handle different camera viewpoint angles remains future work.



(a) User fronto-parallel to the image plane. $\theta = 0^\circ$

(b) viewpoint angle $\theta = -45^\circ$

(c) viewpoint angle $\theta = -75^\circ$

(d) viewpoint angle $\theta = +90^\circ$

Figure 5.5. User performing gestures under various camera viewpoints.

Right now we have preliminary results for the cases where $\theta \in \{\pm 45^\circ, \pm 75^\circ\}$. First we tested our gesture recognition system from Chapter 4. Recognition rates are reduced drastically down to 23.33% with a change in viewpoint of about $+75^\circ$. However when testing our new view invariant method recognition rates remain very high 93.33% proving that our system is indeed viewpoint invariant. Results are depicted in Figure 5.6. Cumulative recognition rate for our proposed method is 98.33% while for our competitor recognition accuracy is reduced down to 41.66%.

Total processing time for the recognition algorithm, including fitting the plane, transforming and normalizing the $3D$ points, on a PC with a 2.00 GHz Intel(R) Xeon(R) E5406 processor, is about 19 seconds. Estimating the plane and transforming takes about 0.16 seconds and normalization takes about 0.15 seconds. The code is rather unoptimized and

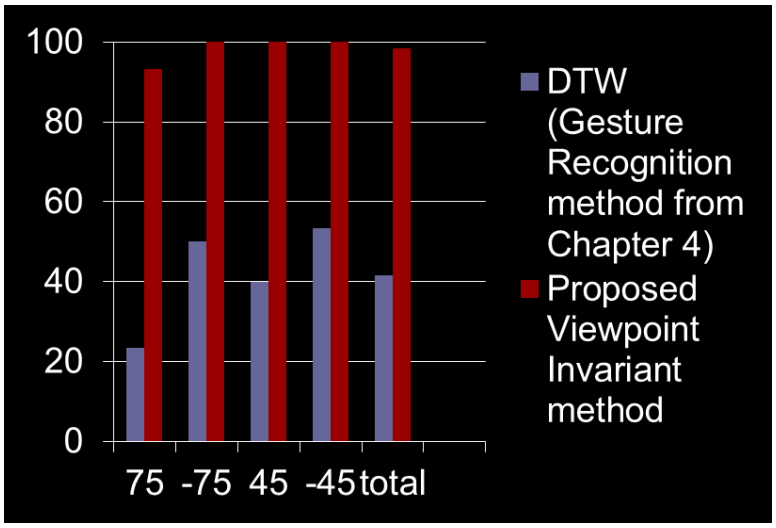


Figure 5.6. Results for our view invariant gesture recognition method. For comparison we tested our gesture recognition method from previous Chapter 4.

implemented in MATLAB R2012a. The plane estimation step takes as input the whole set of $3D$ gesturing points. This introduces a small latency to our recognition algorithm which requires that the whole gesture has to be first captured.

5.5 Discussion and Future Work

This chapter proposed a novel view invariant gesture recognition method that can recognize gestures from videos captured under various camera viewpoints ranging from -75° to $+75^{\circ}$. The proposed system can be trained with gestures captured from a specific viewpoint and tested with gestures from various viewpoints. Our system is all-together translation, scale and viewpoint invariant. To the best of our knowledge few gesture recognition methods satisfy all these three properties at the same time.

Experimental results that the proposed method is indeed view invariant even in the cases of extreme viewpoint angles like $\theta = 75^{\circ}$ or $\theta = -75^{\circ}$. An open research problem that needs investigation is automating temporal segmentation. In the current approach the beginning and end frame for each gesture is manually annotated. Automating this pro-

cedure (e.g., by using a distinct pose for the non-gesturing hand) remains future work. Automatic hand tracking in order to handle different camera viewpoint angles also remains future work. Finally, we would like to expand our gesture recognition system in order to accommodate more challenging gestures from other domains such as the American Sign Language.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

6.1 Discussion of Contributions

This thesis investigated methods for viewpoint invariant gesture recognition and $3D$ hand pose estimation. First, Chapter 2 proposed a viewpoint invariant hand pose estimation method using RGB-D. It proposed an exemplar-based method that relies on similarity measures employing depth information. At the same time, towards making 3D hand pose estimation methods more automatic, a novel hand segmentation method has been introduced which also relies on depth data. Contrary to previous approaches the proposed clutter-tolerant method is all-together: user-independent, automatically detects and segments the hand from a single image (no multi-view or motion cues employed) and provides estimation not only for the 3D pose orientation but also for the full hand articulation parameters. Depth information increases the discrimination power of our method, according to the experiments conducted. At the same time, differences in anthropometric parameters and clutter in background are two important factors influencing recognition accuracy of our system. Experimental evaluation of these two factors has been provided by quantitatively measuring their influence on the performance of our proposed similarity measures. Estimating hand pose from a single image can be useful in automatically initializing hand trackers that can be integrated with gesture recognition systems.

On the topic of gesture recognition, a novel method is proposed that combines a well known similarity measure, namely the Dynamic Time Warping (DTW), with a new hand tracking method which on based on motion from frame differencing which we combine with a depth segmentation according to the depth information we have for each pixel.

Depth frames have been captured using Microsoft's Kinect™ RGB-Depth sensor. First, at Chapter 3 we evaluate our depth hand tracker against one popular open source user skeleton tracker by examining its performance on random signs from a dataset of American Sign Language (ASL) signs. This evaluation can be seen as a contribution since it can serve as a benchmark for the assessment of more advanced detection and tracking methods that utilize RGB-D data. Another contribution of Chapter 3 is the introduction of a structured motion dataset of (ASL) signs which has been captured in both RGB and depth format using a Microsoft Kinect™ sensor and it will enable researchers to explore body part (i.e., hands) detection and tracking methods, as well as gesture recognition algorithms.

Chapter 4 proposes a novel gesture recognition system that integrates the depth hand tracker presented at chapter 3. The proposed method was one of the earliest ones that used depth information from the Kinect™ sensor. Some interesting properties of the proposed system are the following:

- It performs very well even in very challenging environments with the presence of multiple "distractors" like moving objects, or skin colored objects (e.g., face, non-gesturing hand, background objects).
- It is robust to overlaps between the gesturing hand and the face.
- It is translation and scale invariant; the gesture can occur in any part of the image.
- Unlike HMMs and CONDENSATION-based gesture recognition our method requires no knowledge of observation and transition densities, and therefore can be applied even if we have a single example per class.
- Our method can be generalized and applied to recognize a wider range of gestures, other than signs of digits.

Finally, Chapter 5 contributes to the state of the art by extending the proposed gesture recognition system in order to handle cases where the user is not standing fronto-parallel with respect to the camera. In our experiments we recorded videos and we managed to

recognize gestures from a set of different viewpoints: $\{\pm 45^{\circ}, \pm 75^{\circ}\}$. Our view invariant gesture recognition method has some attractive properties which can be summarized as follows:

1. It is trained from videos captured under one specific camera viewpoint but it can be tested with gestures captured under arbitrary camera viewpoints. In our experiments we opt to train our system with a camera viewpoint where the user is standing fronto-parallel to the image plane. For testing the videos are captured under the following set of viewpoints $\{\pm 45^{\circ}, \pm 75^{\circ}\}$.
2. It is all-together translation, scale and viewpoint invariant. To the best of our knowledge few gesture recognition methods satisfy all these three properties at the same time
3. It employs an affordable, commercially available sensor (i.e., Microsoft Kinect™) as opposed to an expensive laboratory sensor or a cumbersome calibrated multi-camera set-up

6.2 Future Work

On the topic of $3D$ hand pose estimation the retrieval accuracy for the proposed system is still too low to be used as a stand-alone module for 3D hand pose estimation and/or gesture recognition. Future work will be to define more sophisticated similarity measures further exploiting depth information. At this point depth data and the way we have used it can be regarded as a $2.5D$ source of information. By using the Kinect™ camera's intrinsic and extrinsic parameters we can construct $3D$ point clouds and start exploiting this richer source of information. We could experiment with features like surface normals and other $3D$ feature descriptors. Our system currently doesn't achieve real-time

performance. In order to do so and since our method is inherently parallel, additional future work will be to take advantage of the GPU's parallel processing power.

On the topic of view invariant gesture recognition an open problem that needs investigation is automating temporal segmentation. In the current approach the beginning and end frame for each gesture is manually annotated. Automating this procedure (e.g., by using a distinct pose for the non-gesturing hand) remains future work. Automatic hand tracking in order to handle different camera viewpoint angles also remains future work. Finally, we would like to expand our gesture recognition system in order to accommodate more challenging gestures from other domains such as the American Sign Language.

REFERENCES

[1] Smith Micro, Aliso Viejo, CA, "Poser 8," http://poser.smithmicro.com/poser.html.

[2] P. Doliotis, A. Stefan, C. McMurrough, D. Eckhard, and V. Athitsos, "Comparing gesture recognition accuracy using color and depth information," in *International Conference on PErvasive Technologies Related to Assistive Environments*, 2011.

[3] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff, "Simultaneous localization and recognition of dynamic hand gestures," in *IEEE Motion Workshop*, 2005, pp. 254–260.

[4] V. Athitsos and S. Sclaroff, "Estimating 3d hand pose from a cluttered image," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2003, pp. II–432–9 vol.2.

[5] Microsoft Corp. Redmond WA., "Kinect Xbox 360," http://www.xbox.com/kinect.

[6] M. Schneider and C. Stevens, "Development and testing of a new magnetic-tracking device for image guidance," *Proceedings of SPIE*, vol. 7035, no. 1, pp. 65 090I–65 090I–11, 2007.

[7] R. Y. Wang and J. Popović, "Real-time hand-tracking with a color glove," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 63:1–63:8, July 2009.

[8] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly, "Vision-based hand pose estimation: A review," *Computer Vision and Image Understanding*, vol. 108, no. 1-2, pp. 52–73, 2007.

[9] B. Moghaddam and A. Pentland, "Probabilistic visual learning for object detection," MIT, Tech. Rep. 326, June 1995.

[10] J. Triesch and C. von der Malsburg, "Robotic gesture recognition," in *Gesture Workshop*, 1997, pp. 233–244.

[11] W. Freeman and M. Roth, "Computer vision for computer games," in *IEEE International Conference on Automatic Face and Gesture Recognition*, 1996, pp. 100–105.

[12] Y. Wu and T. Huang, "View-independent recognition of hand postures," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2000, pp. 88–94.

[13] I. Oikonomidis, N. Kyriazis, and A. Argyros, "Efficient model-based 3d tracking of hand articulations using kinect," in *British Machine Vision Conference*, 2011.

[14] M. de La Gorce, D. J. Fleet, and N. Paragios, "Model-based 3d hand pose estimation from monocular video," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 9, pp. 1793–1805, 2011.

[15] I. Oikonomidis, N. Kyriazis, and A. A. Argyros, "Markerless and Efficient 26-DOF Hand Pose Recovery," in *Asian Conference on Computer Vision*, Queenstown, New Zealand, 2010.

[16] J. Rehg and T. Kanade, "Model-based tracking of self-occluding articulated objects," *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 0, p. 612, 1995.

[17] C. Keskin, F. Kiraç, Y. E. Kara, and L. Akarun, "Real time hand pose estimation using depth sensors," in *IEEE International Conference on Computer Vision Workshops*, 2011, pp. 1228–1234.

[18] N. Pugeault and R. Bowden, "Spelling it out: Real-time asl fingerspelling recognition," in *IEEE International Conference on Computer Vision Workshops*, 2011, pp. 1114–1119.

[19] Z. Mo and U. Neumann, "Real-time hand pose recognition using low-resolution depth images," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 1499–1505.

[20] P. Doliotis, V. Athitsos, D. I. Kosmopoulos, and S. J. Perantonis, "Hand shape and 3d pose estimation using depth data from a single cluttered frame." in *International Symposium on Visual Computing*, vol. 7431, 2012, pp. 148–158.

[21] L. G. Khachiyan, “Rounding of polytopes in the real number model of computation,” *Math. Oper. Res.*, vol. 21, no. 2, pp. 307–320, May 1996.

[22] Minimum Volume Enclosing Ellipsoid, “Matlab Central,” http://www.mathworks.com/matlabcentral/fileexchange/9542-minimum-volume-enclosing-ellipsoid.

[23] Smoothing 2D Contours Using Local Regression Lines, “Matlab Central,” http://www.mathworks.com/matlabcentral/fileexchange/30793-smoothing-2d-contours-using-local-regression-lines.

[24] Moore Neighbor Tracing Algorithm (description), http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/alg.html.

[25] Moore Neighbor Tracing Algorithm (implementation), “Matlab Central,” http://www.mathworks.com/matlabcentral/fileexchange/27639-boundary-tracing-using-the-moore-neighbourhood.

[26] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, “Parametric correspondence and chamfer matching: two new techniques for image matching,” in *In International Joint Conference on Artificial Intelligence*, ser. IJCAI’77, 1977, pp. 659–663.

[27] V. Athitsos and S. Sclaroff, “An appearance-based framework for 3D hand shape classification and camera viewpoint estimation,” in *IEEE International Conference on Automatic Face and Gesture Recognition*, 2002.

[28] OpenNI, “Openni website.” [Online]. Available: http://www.openni.org/

[29] V. Athitsos, C. Neidle, S. Sclaroff, J. Nash, A. Stefan, and A. Thangali, “The american sign language lexicon video dataset,” in *IEEE Workshop on Computer Vision and Pattern Recognition for Human Communicative Behavior Analysis (CVPR4HB)*, June 2008, pp. 1–8.

[30] C. Valli, Ed., *The Gallaudet Dictionary of American Sign Language*. Washington, DC: Gallaudet U. Press, 2006.

[31] I. Guyon, V. Athitsos, P. Jangyodsuk, B. Hamner, and H. Escalante, "Chalearn gesture challenge: Design and first results," in *IEEE Computer Vision and Pattern Recognition Workshops*, 2012, pp. 1–6.

[32] C. Conly, P. Doliotis, P. Jangyodsuk, R. Alonzo, and V. Athitsos, "Toward a 3d body part detection video dataset and hand tracking benchmark," in *International Conference on PErvasive Technologies Related to Assistive Environments*, 2013, pp. 2:1–2:6.

[33] "Developer SDK, toolkit & documentation | kinect for windows," http://www.microsoft.com/en-us/kinectforwindows/develop/. [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/develop/

[34] W. Freeman, "Computer vision for television and games," in *Recognition, Analysis and Tracking of Faces and Gestures in Real-time Systems (RATFG-RTS)*, 1999, p. 118.

[35] T. Starner and A. Pentland, "Real-time american sign language recognition from video using hidden markov models," in *IEEE International Symposium on Computer Vision*, 1995, pp. 265–270.

[36] F. Chen, C. Fu, and C. Huang, "Hand gesture recognition using a real-time tracking method and Hidden Markov Models," *Image and Video Computing*, vol. 21, no. 8, pp. 745–758, August 2003.

[37] J. Martin, V. Devin, and J. Crowley, "Active hand tracking," in *IEEE International Conference on Automatic Face and Gesture Recognition*, 1998, pp. 573–578.

[38] E. Keogh, "Exact indexing of dynamic time warping," in *International Conference on Very Large Databases (VLDB)*, 2002, pp. 406–417.

[39] M. Black and A. Jepson, "Recognizing temporal trajectories using the condensation algorithm," in *IEEE International Conference on Automatic Face and Gesture Recognition*, 1998, pp. 16–21.

[40] M. Isard and A. Blake, "ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework," in *European Conference on Computer Vision (ECCV)*, 1998, pp. 893–908.

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

[42] S. N. P. Vitaladevuni, V. Kellokumpu, and L. S. Davis, "Action recognition using ballistic dynamics." in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[43] R. Messing, C. Pal, and H. Kautz, "Activity recognition using the velocity histories of tracked keypoints," in *IEEE International Conference on Computer Vision*, 2009, pp. 104–111.

[44] A. Wedel, T. Brox, T. Vaudrey, C. Rabe, U. Franke, and D. Cremers, "Stereoscopic scene flow computation for 3d motion understanding," *International Journal of Computer Vision*, vol. 95, no. 1, pp. 29–51–, 2011.

[45] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 886–893 vol. 1.

[46] S. R. Fanello, I. Gori, G. Metta, and F. Odone, "Keep it simple and sparse: Real-time action recognition," *Journal of Machine Learning Research*, vol. 14, pp. 2617–2640, 2013.

[47] R. Fergus, P. Perona, and A. Zisserman, "A sparse object category model for efficient learning and exhaustive recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 380–387 vol. 1.

[48] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *In Workshop on Statistical Learning in Computer Vision, ECCV*, 2004, pp. 1–22.

[49] J. Niebles, H. Wang, and L. Fei-Fei, "Unsupervised learning of human action categories using spatial-temporal words," *International Journal of Computer Vision*, vol. 79, no. 3, pp. 299–318–, 2008.

[50] N. Dardas and N. D. Georganas, "Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques," *IEEE Transactions on Instrumentation and Measurements*, vol. 60, no. 11, pp. 3592–3607, 2011.

[51] D. G. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*, 1999, pp. 1150–1157 vol.2.

[52] M. Ahmad and S.-W. Lee, "Hmm-based human action recognition using multiview image sequences," in *International Conference on Pattern Recognition*, vol. 1, 2006, pp. 263–266.

[53] P. Raamana, D. Grest, and V. Krueger, "Human action recognition in table-top scenarios : An hmm-based analysis to optimize the performance," in *International Conference on Computer Analysis of Images and Patterns*, 2007, vol. 4673, pp. 101–108–.

[54] Q. Shi, L. Wang, L. Cheng, and A. Smola, "Discriminative human action segmentation and recognition using semi-markov model," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[55] L. E. S. Hctor Hugo Avils-Arriaga, "Dynamic bayesian networks for visual recognition of dynamic gestures," *Journal of Intelligent and Fuzzy Systems*, vol. 12, no. 3, pp. 243–250, Jan. 2002.

[56] W.-H. Wang and C.-L. Tung, “Dynamic hand gesture recognition using hierarchical dynamic bayesian networks through low-level image processing,” in *International Conference on Machine Learning and Cybernetics*, vol. 6, 2008, pp. 3247–3253.

[57] D.-Y. Huang, W.-C. Hu, and S.-H. Chang, “Vision-based hand gesture recognition using pca+gabor filters and svm,” in *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2009, pp. 1–4.

[58] D. L. Vail and C. Guestrin, “Conditional random fields for activity recognition,” in *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007, pp. 1–8.

[59] S. P. Chatzis, D. I. Kosmopoulos, and P. Doliotis, “A conditional random field-based model for joint sequence segmentation and classification,” *Pattern Recogntion*, vol. 46, no. 6, pp. 1569–1578, June 2013.

[60] S. B. Wang, A. Quattoni, L. Morency, D. Demirdjian, and T. Darrell, “Hidden conditional random fields for gesture recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 1521–1527.

[61] J. W. Davis and A. F. Bobick, “The representation and recognition of human movement using temporal templates,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 1997, pp. 928–934.

[62] T. Xiang and S. Gong, “Beyond tracking: modelling activity and understanding behaviour,” *International Journal of Computer Vision*, vol. 67, p. 2006, 2006.

[63] B. Ni, G. Wang, and P. Moulin, “Rgbd-hudaact: A color-depth video database for human daily activity recognition,” in *IEEE International Conference on Computer Vision Workshops*, 2011, pp. 1147–1153.

[64] D. Kosmopoulos, P. Doliotis, V. Athitsos, and I. Maglogiannis, “Fusion of color and depth video for human behavior recognition in an assistive environment,” in *International Conference on Human-Computer Interaction*, 2013, vol. 8028, pp. 42–51.

[65] A. Corradini, "Dynamic time warping for off-line recognition of a small gesture vocabulary," in *Recognition, Analysis and Tracking of Faces and Gestures in Real-time Systems (RATFG-RTS)*, 2001, pp. 82–89.

[66] T. Darrell and A. Pentland, "Space-time gestures," in *IEEE Conference on Computer Vision and Pattern Recognition*, 1993, pp. 335–340.

[67] C. Rasmussen and G. Hager, "Probabilistic data association methods for tracking complex visual objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 23, no. 6, pp. 560–576, June 2001.

[68] Y. Sato and T. Kobayashi, "Extension of hidden markov models to deal with multiple candidates of observations and its application to mobile-robot-oriented gesture recognition," in *International Conference on Pattern Recognition (ICPR)*, vol. 2, 2002, pp. 515–519.

[69] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34(1), 1978, pp. 43–49.

[70] M. Jones and J. Rehg, "Statistical color models with application to skin detection," *International Journal of Computer Vision (IJCV)*, vol. 46, no. 1, pp. 81–96, January 2002.

[71] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff, "A unified framework for gesture recognition and spatiotemporal gesture segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 9, pp. 1685–1699, 2009.

[72] H. A. Rowley, S. Member, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions On Pattern Analysis and Machine intelligence*, vol. 20, pp. 23–38, 1998.

[73] D. Weinland, R. Ronfard, and E. Boyer, “Free viewpoint action recognition using motion history volumes,” *Computer Vision and Image Understanding*, vol. 104, pp. 249–257, Nov. 2006.

[74] R. Souvenir and J. Babbs, “Learning the viewpoint manifold for action recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–7.

[75] M. Holte, T. Moeslund, and P. Fihl, “View-invariant gesture recognition using 3d optical flow and harmonic motion context,” *Computer Vision and Image Understanding*, vol. 114, no. 12, pp. 1353–1361, Dec. 2010.

[76] D. Weinland, R. Ronfard, and E. Boyer, “A survey of vision-based methods for action representation, segmentation and recognition,” *Computer Vision and Image Understanding*, vol. 115, no. 2, pp. 224–241, Feb. 2011.

[77] 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.

[78] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, June 1981.

BIOGRAPHICAL STATEMENT

Paul Doliotis received his B.S. degree from the Department of Informatics at Athens University of Economics and Business (AUEB), Greece, in 2004. He received his M.S. from the Department of Cultural Technology and Communication at the University of the Aegean, Greece, in 2008.

He received his Ph.D. degree in Computer Science from the Department of Computer Science and Engineering at The University of Texas at Arlington, in 2013. Since February 2013 he has been working within the R&D team of Wynright Robotics as a Computer Vision Engineer. His current research interests lie in the area of Computer Vision, Machine Learning, American Sign Language/Gesture Recognition and 3D Perception.