University of Thessaly

Doctoral Thesis

Low Complexity Hand Gesture Recognition

Author:
Stergios Poularakis

Supervisor:
Dr. Ioannis Katsavounidis

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Department of Electrical and Computer Engineering

November 2014

Declaration of Authorship

I, Stergios Poularakis, declare that this thesis titled, 'Low Complexity Hand Gesture Recognition' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"tempus fugit"

Virgilio, Georgiche 3:284, 29 BC

UNIVERSITY OF THESSALY

Abstract

Department of Electrical and Computer Engineering

Doctor of Philosophy

Low Complexity Hand Gesture Recognition

by Stergios POULARAKIS

Gesture recognition is an expressive, alternative means for Human Computer Interaction (HCI), which recently drew significant attention after the release of mass consumer applications and devices, including gesture–controlled interactive TV systems (iDTV) and advanced video–game environments. In this work, we propose a complete gesture recognition framework for continuous streams of static postures and dynamic trajectories of digits and letters, targeting both high recognition accuracy and increased computational efficiency. Special emphasis is given on four fundamental gesture recognition problems, i.e. hand detection and feature extraction, isolated recognition, gesture verification, and gesture spotting on continuous data streams.

Specifically, we propose a novel finger detection method, based on geometrical hand contour features (apex detection) and show its importance in hand posture recognition. We then present our approach for isolated recognition, which is based on Maximum Cosine Similarity (MCS) and a tree–based fast Nearest Neighbor (fastNN) technique, showing its high recognition accuracy and computational efficiency. Additionally, we relate the computational time required by fastNN for the classification of an unknown query vector to its Mahalanobis distance and maximum cosine similarity with respect to the set of training examples. This property allows us to perform gesture verification, while it significantly reduces the search time.

Finally, we design a complete framework for gesture spotting on continuous streams of hand data, solving the joint problem of both gesture detection and recognition. Specifically, we model subgesture relationships in a probabilistic way, using both the categories and the relative time positions of overlapping gesture candidates. Additionally, we introduce a novel metric of ranking conflicting gesture candidates, based on their time duration and cosine similarity score, which offers high conflict resolution results for sequences of digits and letters.

In all cases, we support our arguments through thorough experiments on real and synthetic gesture datasets, as well as with real–time gesture spotting applications.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Περίληψη

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διδακτορικό Δίπλωμα

Σύστημα Αναγνώρισης Χειρονομιών Χαμηλής Υπολογιστικής Πολυπλοκότητας

Στέργιος Πουλαράκης

Η αναγνώριση χειρονομιών αποτελεί ένα εκφραστικότερο εναλλακτικό μέσο Επικοινωνίας Ανθρώπου Μηχανής (Human Computer Interaction), το οποίο έλαβε ιδιαίτερη σημασία μετά την εμφάνιση μαζικών εφαρμογών και συσκευών, όπως συστημάτων αλληλεπιδραστικής τηλεόρασης (iDTV) και προηγμένων συσκευών ηλεκτρονικού παιχνιδιού.

Στην παρούσα εργασία προτείνουμε ένα πλήρες σύστημα αναγνώρισης χειρονομιών αριθμητικών ψηφίων και γραμμάτων, στοχεύοντας τόσο σε υψηλά ποσοστά επιτυχούς αναγνώρισης όσο και σε χαμηλή υπολογιστική πολυπλοκότητα. Ξεχωριστή έμφαση δίνεται σε τέσσερα σημαντικά προβλήματα: 1) ανίχνευση χεριού και εξαγωγή χαρακτηριστικών, 2) αναγνώριση απομονωμένων χειρονομιών, 3) επιβεβαίωση χειρονομιών και 4) ανίχνευση και αναγνώριση χειρονομιών σε συνεχείς ροές δεδομένων.

Ειδικότερα, προτείνουμε μια νέα μέθοδο ανίχνευσης των δακτύλων του χεριού, βασιζόμενοι σε γεωμετρικά χαρακτηριστικά της περιφέρειας (ανίχνευση γωνιών) και δείχνουμε την σημασία της στην αναγνώριση στατικών χειρομορφών.

Έπειτα, παρουσιάζουμε την προσέγγισή μας για αναγνώριση απομονωμένων χειρονομιών αριθμητικών ψηφίων και γραμμάτων, η οποία βασίζεται στην Μέγιστη Ομοιότητα Συνημίτονου (Maximum Cosine Similarity) και σε μια γρήγορη δενδρική τεχνική αναζήτησης διανυσμάτων (fast Nearest Neighbor – fastNN), η οποία παρουσιάζει τόσο υψηλά ποσοστά αναγνώρισης όσο και αυξημένη υπολογιστική αποδοτικότητα. Επιπρόσθετα, συσχετίζουμε την υπολογιστική πολυπλοκότητα αναγνώρισης ενός διανύσματος ερώτησης με την Mahalanobis απόστασή του από το σύνολο των δεδομένων εκπαίδευσης και την ομοιότητα συνημιτόνου. Η ιδιότητα αυτή μας επιτρέπει να πετύχουμε επιβεβαίωση χειρονομιών, ενώ παράλληλα μειώνει τον υπολογιστικό χρόνο κατηγοριοποίησης.

Τέλος, προτείνουμε ένα πλήρες σύστημα αναγνώρισης χειρονομιών σε συνεχείς ροές δεδομένων, λύνοντας το διπλό πρόβλημα του εντοπισμού και αναγνώρισης χειρονομιών. Συγκεκριμένα, μοντελοποιούμε πιθανοτικά τις σχέσεις μεταξύ αλληλοσυγκρουόμενων υποψήφιων χειρονομιών, χρησιμοποιώντας τόσο τις κατηγορίες τους όσο και τις μεταξύ

τους χρονικὲς εμφανὶσεις. Επιπρὸσθετα, εισὰγουμε μια νὲα μὲθοδο κατὰταξης αλληλοσυγκρουὸμενων υποψὴφιων χειρονομιὼν, προτεὶνοντας μια μετρικὴ που συνδυὰζει τὸσο την ομοιὸτητα συνημὶτονου τους ὸσο και την χρονικὴ τους διὰρκεια, οδηγὼντας σε υψηλὰ ποσοστὰ επιτυχοὺς αναγνὼρισης σε ακολουθὶες αριθμητικὼν ψηφὶων και γραμμὰτων.

Σε ὸλες τις περιπτὼσεις, υποστηρὶζουμε τα επιχειρὴματὰ μας μὲσω εκτεταμὲνων πειραμὰτων σε πραγματικὲς και συνθετικὲς βὰσεις δεδομὲνων χειρονομιὼν, καθὼς και μὲσω ανὰπτυξης εφαρμογὼν πραγματικοὺ χρὸνου.

Acknowledgements

At this point, where every technical detail of this thesis has been described extensively in a formal way – cold and objective, as it ought to be –, I feel glad adding a few more emotional lines, expressing my gratitude to those people who truly helped me in this long journey through – and towards – knowledge.

First of all, I would like to thank my supervisor, Professor Ioannis Katsavounidis, for his continuous trust, encouragement and support during the years of my graduate studies. His brilliant way of problem analysis and solving through constructive discussion, critical observations and thorough experimentation, as well as his solid faith in scientific principles and ethics were inspiring and encouraging, making the learning process exciting and the learning product quite fulfilling. Our collaboration brings me only good memories in mind and I hope that it has truly met the expectations set from the beginning.

I would also like to thank Professors Elias Houstis and Aikaterini Housti for participating in the thesis committee, as well as for the valuable lessons of engineering and ethics they provided in and out of classroom and their continuous support during my ten years of studies at the University of Thessaly.

I would also like to thank Professor Gerasimos Potamianos for the excellent collaboration in classroom as well as for the long discussions we had on several aspects of pattern recognition and signal processing.

I would also like to thank Professors Panos Tsakalides and Athanasios Mouchtaris for their constructive questions and comments during the examining process, as well as for the beautiful collaboration we had in the "Efficient Location-Aware Audio-Visual Delivery of High-Quality Content to Mobile Devices (AVID-MODE)", PEOPLE–IAPP, FP7, research project, which partially funded my graduate studies.

I would also like to thank Professor Leontios Hadjileontiadis for participating in the examining committee, further enriching the examining process with constructive questions and interesting ideas for future work.

Special thanks go to Dr. Alexia Briassouli who inspired me to love research as an undergraduate and graduate student, while she kindly continued participating in short discussions during the rest years of my studies.

Special thanks also go to Prof. Sigal Berman and Dr. Darya Frolova for kindly providing access to their Kinect gesture dataset as well as for their willingness to discuss some of the related results.

I am also grateful to my closest friends and colleagues for helping me release the tension and deal with anxiety and frustration after a tiring day in the lab.

Last but not least, I would like to thank the members of my family, literally for everything they offered me in all aspects of life. They are the people I love the most, and this thesis is dedicated to them with all my heart.

Contents

List of Figures

List of Tables

Abbreviations

CRF	**C**onditional **R**andom **F**ields
DTW	**D**ynamic **T**ime **W**arping
EER	**E**qual **E**rror **R**ate
fastNN	**f**ast **N**earest **N**eighbor
FAI	**F**ull **A**ction **I**nstances
FD	**F**ourier **D**escriptor
HMM	**H**idden **M**arkov **M**odels
KNN	**K** **N**earest **N**eighbor
LCS	**L**ongest **C**ommon **S**ubsequence
MCS	**M**aximum **C**osine **S**imilarity
ROC	**R**eceiver **O**perating **C**haracteristic
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SSE2	**S**treaming **S**IMD **E**xtensions
SVM	**S**upport **V**ector **M**achines

To my family

Part I

Introduction

Chapter 1

Introduction

Contents

Gesture recognition is an expressive, alternative means for Human Computer Interaction (HCI). The recent release of mass consumer applications and devices, including gesture–controlled interactive TV systems (iDTV) and advanced video–game environments, increased significantly the interest in gesture recognition technology. In this chapter, we introduce the basic terminology, along with some appropriate definitions, which will be used throughout the rest of this work.

1.1 Definition and types of gestures

Gestures are expressive, meaningful body motions involving physical movements of the fingers, hands, arms, head, face or body, with the intent of conveying meaningful information or interacting with the environment [1]. Intention for interaction is the key feature that discriminates gestures from general *human motion* or *human activities*, such as walking and running. Moreover, human activities typically contain many repetitions of a basic pattern, known as *action cycle*, such as a step in walking or a jump in jumping. On the contrary, gestures usually last for a shorter amount of time and typically one repetition is performed.

In this work, we mainly focus on hand gestures, i.e. gestures performed by arms, palms and fingers, and commonly refer to them as "gestures". Gestures can be *static postures*, when the hand does not move and most of the information lies in finger configuration, or *dynamic trajectories*, when information lies on hand's motion. Some gestures may have both static and dynamic elements, such as some signs of sign languages.

1.2 Gesture recognition

Gesture recognition refers to automatic detection and classification of gestures, using computational power. Typically, user's behaviour is continuously recorded by some capturing device, such as standard 2D cameras, depth or infra–red (IR) cameras, touch screens and body sensors. This raw input signal is subsequently processed to isolate hand(s) and any other meaningful information from the rest of the scene. Typically, background subtraction techniques, combined with face and skin detectors, may be involved in this low–level processing step.

Mid–level processing involves *feature extraction*, i.e. extraction of *discriminating features* which uniquely characterize a gesture class. Such features may describe hand's shape, location and trajectory, or fingers' articulations, and typically result in a multi-dimensional time series.

Such time series is processed in a continuous and systematic fashion, in order to spot and recognize meaningful patterns (*gesture spotting*). A finite vocabulary of predefined gestures defines all valid gestures and their characteristics, as derived from some training phase. After a gesture is unambiguously spotted, system's Artificial Intelligence (AI) component is responsible for planning and taking some action, which forms system's response in the Human–Computer dialogue.

In general, gestures may have more than one possible interpretation, based on environment conditions or cultural differences [1]. Additionally, different users may be assigned different gesture–to–interpretations mappings, based on ranking scales, criteria, identity, etc. Such issues should be taken into account when designing a gesture–based application.

1.3 Gesture spotting

Gesture spotting refers to gesture recognition on continuous motion streams, which poses the joint problem of estimating both the category of the performed gesture, as well as its

(a) (b) (c) d)

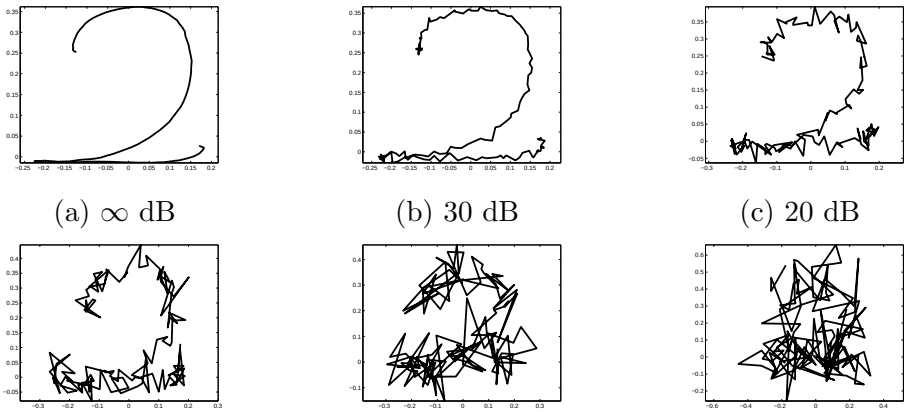
FIGURE 1.1: (a) A typical instance of the subgesture problem: recognition of digit "8" includes recognition of digit "5" too. (b) Confusion after recognition of digit "4" (solid line). Does the dotted line correspond to digit "1", "7" or a downwards movement? (c) Confusing digit "1" to "7" after recognition of digit "0". (d) Avoiding confusion due to different "7" shape in Digits6D datastream [3].

duration and time boundaries. Typically, all gesture spotting techniques need to deal with two fundamental problems [1]: 1) *spatio-temporal variability*, i.e. a user cannot reproduce the same gesture at the exact same shape and duration, and 2) *segmentation ambiguity*, i.e. problems caused by erroneous time boundary detection.

Specifically, some gestures may be *subgestures* of other gestures (e.g. the digits "5" and "8" in Fig. 1.1–a) [2]. Additionally, connectors between two gestures may be incorrectly perceived as a gesture. For example, the last dotted segment in Fig. 1.1–b can be mistaken to the digit "1" rather than a downward connecting segment to a subsequent gesture.

Necessity of spotting is also applicable in the case of static postures. Typically, a person raises his hand, holds it on the air for a while, and then lowers it down. This relatively simple scenario produces a lot of video frames (or other sensor data). Intuitively, one can expect that only a few video frames are critical, when the hand is almost still on the air (*posture locking* phase). Once again, the continuous nature of input data makes recognition a much more challenging task and thus spotting is necessary.

Recognition of isolated gestures (*isolated recognition*) is a much easier problem and attracted the interest of many early research works. In this scenario, the input signal contains only one complete gesture (e.g. a trajectory or a posture) and thus all available data may be used for recognition. This optimistic scenario occurs in touch screens, where a pressure signal is generated whenever our fingers touch the screen. However, when cameras and other capturing devices are used, one may either try to detect the time boundaries of the performed gestures and then apply standard isolated recognition techniques (*direct approach*), or may form joint hypotheses about both boundaries and categories of gestures (*indirect approach*).



(a) ∞ dB (b) 30 dB (c) 20 dB

(d) 15 dB (d) 10 dB (d) 5 dB

FIGURE 1.2: (a) - (d) Digit "2" from 6DMG dataset [3], after adding Gaussian noise at various SNRs.

1.4 Gesture verification

Gesture verification refers to verifying whether a spotted gesture is indeed an instance from a predefined vocabulary or an invalid gesture (*false positive*). In this work, we consider three main types of invalid gestures, namely 1) *out–of–vocabulary* gestures, 2) *noisy* gestures and 3) *random* movements.

As out–of–vocabulary gestures, we consider shapes and symbols that may have some meaning for humans, but are not included in the system's vocabulary. For example, a system designed to recognize the ten Hindu–Arabic digits (0-9) should reject all the twenty–six Latin letters (A–Z, a–z), mathematical symbols, etc.

Noisy gestures refers to valid gestures with some additive noise. Noise may always appear in the gesture data due to device flaws (e.g. low thermal noise), user's inexperience with the recognition system or user's temporal or permanent behavioural characteristics, such as anxiety, trembling hand and movement limitations. As Fig. 1.2 shows, a desirable property for a recognition system would be to allow some low noise and reject highly noisy gestures.

Finally, completely random movements can be used to model uncontrolled scenarios during system's inactivity periods, i.e. when users don't intend to interact with it. For example, considering a gesture–based interactive TV system, users may produce accidental signals while drinking water or walking in front of the camera.

1.5 Contributions of this thesis

In this thesis, we design a complete gesture recognition system, operating on continuous streams of hand data (hand coordinates, hand shapes) coming from some capturing device (2D/depth camera, accelerometers), as shown in Fig. 1.3. Data are segmented into multiple overlapping chunks and go through an *isolated recognition* module, which assigns them with a candidate class label. Additionally, a *gesture verification* module decides whether the candidate gesture is valid or invalid, using the results of isolated recognition and gesture characteristics (confidence score of classifier, *difficulty* of classification, gesture time duration, etc.). Successful candidates are added in a list of possible gestures (*conflicting gestures*), which are typically overlapping with each other and thus we need to decide which one is the real gesture signed by the user. This is the responsibility of the *gesture spotting* module, which applies *conflict resolution* techniques, leading to the final spotting result.

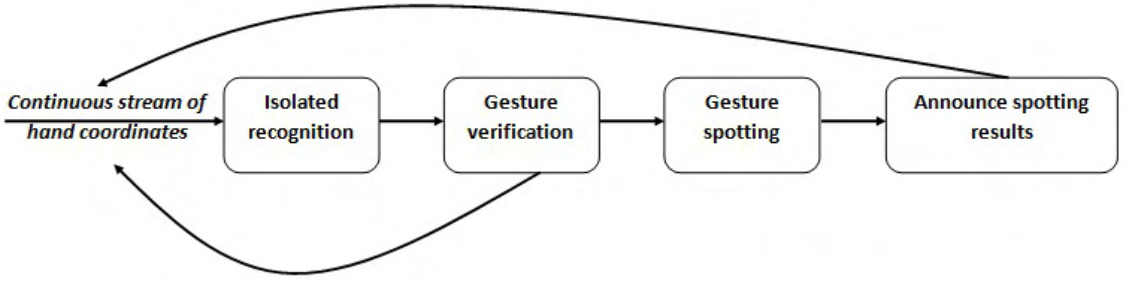


FIGURE 1.3: General structure of our proposed approach.

In short, the main contributions of this thesis are:

- Proposing a complete framework for hand gesture recognition of digits and letters based on Maximum Cosine Similarity (MCS) and a tree–based fast Nearest Neighbor technique (fastNN). Our approach is simple, accurate, low–complexity, robust against noise and limited number of training examples, and can be implemented very efficiently in embedded platforms. Moreover, it is not bounded to a specific hand detection technique, since it can be successfully used with data from a regular video camera, a Microsoft (R) Kinect^{TM} device or a hand held sensor device, such as Nintendo (R) Wii remote^{TM} .

- Studying analytically all four main gesture recognition problems, i.e. feature extraction, isolated gesture recognition, gesture verification and gesture spotting, providing links between accuracy and computational efficiency.

- Relating the probability of inlier of a query gesture with the computational time needed to be classified by fastNN. Moreover, we explore additional parameters, such as dimensionality and number of training examples.

- Proposing a novel method to perform conflict resolution, combining both the cosine similarity score (measuring the quality of recognition) and the time duration of the candidate gestures.
- Introducing the usage of time boundaries in a probabilistic framework for learning subgesture relationships.
- Proposing a novel approach for finger detection, showing its usefulness in hand posture recognition.
- Exploring information–theoretic measures to evaluate gesture recognition and verification performance.

In all cases, we conduct thorough experiments on real and synthetic gesture datasets.

Chapter 2

Literature Review

Contents

In the last twenty years, many research works have investigated gesture recognition for a variety of applications. Human Computer Interaction (HCI) seems to be the most common target field, promising efficient interaction to people with visual and hearing [4] disabilities, as well as increased naturalness in interactivity to typical users. A lot of works tried to recognize sign language gestures, involving both static alphabet postures as well as dynamic word gestures and whole sentences [5]. Many other works investigated recognition of dynamic trajectories, such as command strokes, digits and letters [2, 6–11], while others worked on recognition of hand shapes, involving digits and simple mathematical operations [12–17]. Recently, several international challenges were organized, targeting recognition of generic body gestures, where not only hands, but also face and body pose are important for recognition [18, 19].

In this chapter, we review some of the most important recent approaches, involving recognition of dynamic trajectories, static postures, sign languages and body gestures. We first provide a general overview of sensors, feature extraction methods and classifiers used by most approaches, after which we review each gesture type separately. Special emphasis is given on gesture spotting in continuous data streams, which is one of the most challenging gesture recognition tasks and an important contribution of this work.

2.1 Sensors, features and classification methods

Selecting the gesture capturing device is an important aspect of a gesture recognition system, affecting the interface of human–computer interaction, power consumption and recognition accuracy. Standard 2D cameras are probably the most commonly used devices, mainly due to their simplicity, low cost, integration in modern mobile devices and development of efficient image processing libraries [20]. Recently, the so called *RGB–D* sensors, such as Microsoft (R) Kinect$^{\text{TM}}$, offer an additional depth component, alleviating a lot of classical Computer Vision problems, such as occlusions and varying lighting conditions [21, 22]. On the other hand, body sensors, such as accelerometers and gyroscopes in mobile phones [23, 24], or forearm surface electromyogram (sEMG) [25–27], provide immediate hand detection results, at the cost of reduced interaction naturalness. For more information regarding sensors for gesture recognition systems, the reader may refer to the survey provided by Berman and Stern [28].

The choice of sensor also affects the types of gestures that may be recognized, as well as the appropriate gesture representation methods. Most body sensor–based systems recognize dynamic trajectories, representing them as time series of hand palm positions and other motion features [23, 29, 30]. However, a few works attempt recognizing sign language postures and dynamic gestures, relying on complex electromyographic gloves [31, 32]. On the other hand, 2D and RGB–D cameras allow for richer vocabularies, involving static postures, dynamic trajectories, signs and body gestures. However, increased flexibility requires more evolved feature extraction methods in order to describe complex hand shapes [12], trajectories and body poses [33].

Since gestures form a time–series of observed features, a lot of state–based sequential models have been applied in the literature, including Hidden Markov Models (HMMs) [30, 34, 35], Dynamic Time Warping (DTW) [2], Longest Common Subsequence (LCS) [10, 11], Conditional Random Fields (CRFs) [36–38], Dynamic Bayesian Networks (DBNs) [39] and Convolutional Neural Networks [40]. Such approaches generally estimate a likelihood score for each gesture category, by summing the likelihood of each observation given the previous observations. The Markov assumption helps reducing the number of training examples, while preserving most of system's recognition ability. Additionally, Viterbi–like [41] dynamic programming techniques deal efficiently with the computation of some *optimal path* across all possible state combinations.

Other approaches treat gestures in a global way, extracting holistic features from the entire time–series, such as downsampled trajectory points [42], bags of visual words [43] and Activity History Images [44]. These global features can then be used with standard

classification methods, such as Nearest Neighbor, Support Vector Machines (SVMs) [45] and Neural Networks [46].

When recognizing static postures, a single video frame containing the final hand shape is usually discriminative enough for accurate recognition. Typically, global or local features, such as Fourier descriptors [17, 47] and Histograms of Oriented Gradients [48, 49] can achieve satisfactory posture recognition results.

Due to the presence of other humans, or skin-like surfaces, standard hand detection methods prove insufficient for robust gesture recognition. In such cases, multiple hand position and transition hypotheses may be considered, while recognition is performed on the path of maximum likelihood [2, 34].

2.2 Gesture spotting

Gesture spotting refers to gesture recognition on continuous motion streams, which poses the joint problem of estimating both the category of the performed gesture, as well as its duration and time boundaries. Spotting is quite challenging due to *spatio-temporal variability* among different users and gesture instantiations, and *segmentation ambiguity* by erroneous time boundary detection and complex subgesture relationships. Gesture spotting methods may be classified as *direct* or *indirect* [2].

Direct approaches first detect the time boundaries of the performed gesture and then apply standard isolated recognition. Typically, motion cues (e.g. velocity) [50–52] or specific start and end marks (e.g. an open/closed palm [15] and short pauses between consecutive gestures [25, 26]) can be employed for time boundary detection.

Zhang et al.[31] used Hidden Markov Models (HMM) with acceleration and surface electromyographic (SEMG) signals to recognize 72 Chinese Sign language words. Spotting was based on thresholding the SEMG signal values. In a similar approach, Lu et al.[32] used Dynamic Time Warping (DTW) and SEMG thresholding to recognize 19 gestures. Jiang et al.[53] required the gesturing hand to return to a *rest* position between consecutive gestures.

In a multi–modal approach, Wan et al.[54] used audio data for spotting, assuming that audio returns to silence between consecutive hand and body gestures. Wang and Chuang [55] used accelerometers on a digital pen and probabilistic neural networks for handwritten digit and gesture recognition. Gesture time boundaries were signalled by the user, through pressure of a button in the pen.

On the other hand, indirect approaches form joint hypotheses about both time boundaries and categories of gestures. Some model–based indirect approaches used HMMs [56, 57], CRFs [36–38] or adaptive Boosting [58] and learned a non–gesture (or garbage) model, detecting gesture boundaries mainly as zero crossings of differential observation likelihood.

Alon et al.[2] created lists of candidate gestures, thresholding the DTW distance scores at continuous time frames. A major novelty of that work was learning and using subgesture relationships for better conflict resolution during the spotting process. Specifically, when a subgesture and its supergesture coexisted in the list of candidates, the supergesture was always favoured and the subgesture removed.

Frolova et al.[10] proposed Most Probable Longest Common Subsequence (MPLCS) for digit gesture recognition. Spotting and gesture verification were based on an evolved thresholding scheme on the candidate probability scores. The authors also explored direct spotting using motion cues for stroke segmentation and a stroke combination method to receive the final recognition results.

Stern et al.[11] modelled gestures as sequences of most discriminating sub–segments (MDS), inspired by the concepts of phonemes and strokes in speech and handwriting recognition, respectively. Their system scans the input stream until a primary MDS is detected, where a candidate gesture is believed to be present. Then, it looks for secondary MDS on adjacent time frames. The final verification and spotting result depends on the appearances of MDS and their Longest Common Subsequence (LCS) scores.

2.3 Recent gesture recognition approaches

2.3.1 Recognizing dynamic trajectories

Approaches featuring dynamic trajectories typically involve vocabularies with gestures "written on the air" by user's hand, such as the 10 Hindu–Arabic digits (0–9), the 26 Latin letters (A–Z, a–z) or various command–like strokes, such as "move to next page" or "turn left". Such vocabularies could be valuable for a variety of applications, including control of entertainment systems by users with finger dysplasia and other hand impairments. In this section we briefly review some of the most important methods found in the literature, using 2D cameras [2, 56, 59], Kinect™ [10, 60], accelerometers [29, 42, 61] and touch screens and hand writing electronic pens [6–9].

In an early computer vision approach, Lee and Kim [56] used Hidden Markov Models (HMM) and a threshold model to control a PowerPoint ™ presentation through 10 command gestures. Their approach achieved highly accurate spotting results and real–time processing on a low–end computational system.

Alon et al.[2] proposed Dynamic Space-Time Warping (DSTW), an extension of Dynamic Time Warping (DTW) [62], which considers multiple hand hypotheses in some very challenging background scenarios. Their system extends Continuous Dynamic Programming (CDP) [63] and achieves high gesture spotting results on their 10–digit dataset. While this approach is very effective, it requires much computational time mainly due to the large number of hand candidates at each frame. For this reason, the same authors applied some pruning process, while Escalera et al.[59] improved their approach by considering only the most probable candidates of each frame.

Frolova et al.[10] extended the Longest Common Subsequence (LCS) algorithm [64] and proposed the Most Probable LCS (MPLCS) method for gesture recognition of digits. MPLCS resembles DTW but can achieve a better alignment by ignoring some trajectory parts and maximizing the length of the common subsequence of two time series. MPLCS achieved high recognition results (98.3%) on a 10–digits dataset obtained from the Kinect device and proved to perform significantly better than HMMs (89.5%) and slightly worse than Conditional Random Fields (99.1%).

Stern et al.[11] modelled gestures as sequences of most discriminating sub–segments (MDS), using LCS to evaluate the intermediate segment results. Their approach performed better than HMMs for 10–digit recognition, achieving 89.6% gesture recognition accuracy on continuous gesture streams.

Jiang et al.[53] defined 8 *minimal effort* dynamic gestures for people with upper extremity physical impairments, using the Borg Scale [65] to rank the physical stress required to perform each gesture. They first applied skin–color detection and depth thresholding to detect user's hands, as well as face, and then tracked them using a 3D particle filter framework. For spotting, hand was required to return to a *rest*, i.e. a predetermined, position, while recognition involved DTW for gesture alignment and the condensation algorithm [66] for classification.

Many approaches involve electronic pens, touch screens and accelerometers of modern smart-phones, and have considered gestures as a set of of straight–line strokes on a real or imaginary surface. The popular classifiers $1 [6], $N [7], Protactor [8] and $N–Protactor [9] down–sampled the gesture trajectory and used NearestNeighbor (NN) with Euclidean Distance (ED) or Cosine Similarity (CS) to recognize uni–strokes and multi–strokes on the screen.

Kratz and Rohs [29] proposed Protractor3D, which uses accelerometers and achieves rotation–invariant gesture recognition. Vatavu [42, 67] explored how down–sampling affects the performance of $1 classifier [6] and concluded that it is almost not significant for ED, Maximum Cosine Similarity (MCS) and DTW. However, DTW presented a linear relation between the sampling rate and the number of gesture categories (vocabulary). Very recently, the same author [42] explored the impact of sampling rate and bit depth on 3D gesture classifiers, targeting devices with limited resources, and concluded that few bits and samples are enough for NN classification of simple shapes.

Chen et al.[30] defined 20 command gestures, drawn with Nintendo (R) Wii^{TM} remote (Wiimote). Recognition was based on Hidden Markov Models and a set of 45 features, involving trajectory coordinates, gesture's duration, velocity, orientation, acceleration and other spatial and motion features. Their experiments revealed that simply using normalized trajectory coordinates results in high recognition accuracy for the user–independent case, while an increase of $4-8\%$ may arise when velocity and other features are included. However, using a larger feature set typically compromises computational efficiency, while it requires more training examples.

Xu et al.[23] adjusted *Micro Electro-Mechanical Systems* (MEMS) accelerometers on user's hand to recognize 7 command gestures (*up*, *down*, *left*, *right*, *tick*, *circle* and *cross*). Gesture boundaries were detected based on coordinate and velocity features, while a gesture were represented as a vector of 8 ternary sign features $(+1, 0, -1)$. Quite interestingly, Nearest Neighbor classification resulted in 95.8% recognition accuracy applying this simple representation on 72 testing sequences of 628 gestures in total.

Recently, some approaches studied fusion of different sensor data. Kosmidou et al.[25, 26] combined acceleration and surface electromyographic (SEMG) signals to recognize 61 Greek Sign Language gestures, separating words with small pauses. Their feature set involved Sample Entropy [25] and weighted Intrinsic–Mode Entropy (wIMEn) [26], while recognition was based on Mahalanobis distance. In a similar work, Lu et al.[32] used DTW with acceleration and SEMG signals to recognize 19 gestures (the 10 digits, 5 command gestures and 4 static postures). Spotting was based on thresholding the SEMG signal values, reporting 89.6% recognition accuracy. Liu et al.[68] used Kinect and a 9D body sensor (measuring 3D acceleration, 3D angular velocity and 3D magnetic field strength) to recognize 5 gestures. Isolated recognition was performed, based on HMMs and DTW. In all cases, fusion resulted in higher recognition accuracy, while HMMs greatly outperformed DTW.

Lian et al.[69] designed a complete gesture–based TV control system, consisting of an ultrasonic distance array and a standard 2D RGB camera. In their work, a specially

designed module recognizes user's state (*Absent, Other Action, Controlling, and Watching*), and activates the computationally cheaper ultrasonic distance array or the more expensive 2D camera. Their experiments reported $35-56\%$ savings in power consumption over straightforward implementations.

Quite recently, user authentication based on dynamic gestures received much attention. Guerra-Casanova et al.[61] used "in-air signatures" for user authentication on mobile phones, supporting 50 users and achieving very low error rate in forge attacks (2.8%) with DTW. In another, Kinect–based approach, Tian et al.[60] used DTW for user verification, supporting 18 users and proving that their system is quite safe under many types of attacks.

2.3.2 Recognizing static postures

In static postures the hand remains still in the air and most of the information lies in finger configuration. While postures are not considered natural enough for some applications, such as TV control [69], their role is fundamental in sign languages, either in static alphabets or in dynamic signs, where hand shape and trajectory are equally important for recognition. In some cases, static postures may be used to signal the beginning or ending of dynamic gestures [15]. A classic survey on hand posture recognition may be found in [70].

In a recent work, Kulshreshth et al.[17] used Fourier Descriptors to process data from a Kinect^{TM} system and recognized the number of fingers in the palm with 90% accuracy. Bagdanov et al.[15] classified a palm as either *open* or *closed*, using SURF features [71] and a non-linear Support Vector Machine, reporting 87.8% accuracy. Kurakin et al.[72] performed recognition of 12 dynamic American Sign Language (ASL) signs, using both shape and motion features. Mihail et al.[73] used two Kinect sensors to recognize 10 postures, using the distribution of hand point positions *(hand's point cloud)* as features. Suryanarayan et al.[74] used the ZCam camera [75] and 3D shape descriptors to recognize 6 postures. Lu et al.[32] used DTW with acceleration and surface electromyographic (SEMG) signals to recognize 19 gestures (the 10 digits, 5 command gestures and 4 static postures).

Recently, some other approaches based on global features and hand models appeared as well. Keskin et al.[16, 76] fit a 3D skeleton to hand points, achieving almost perfect recognition (99.9%) for the 10 ASL digits on synthetic and real datasets. Billiet et al.[77] learned a hand model with 9 degrees of freedom, describing each finger as *stretched* or *closed*, and recognized 8 postures. Oikonomidis et al.[12] tracked the 3D position, orientation and full articulation of a human hand, based on Particle Swarm Optimization

and managed to control computational complexity by utilizing GPUs. Dapogny et al.[78] applied the human pose estimation method of Shotton et al.[79] to learn hand models of musical gestures from depth images. Yao et al.[80] learned a 3D contour model and used prior knowledge of hand structure to improve posture recognition accuracy.

The works of Ren et al.[13] and Doliotis et al.[14] are the most relevant to our work. Doliotis et al.[14] performed hand–palm separation from Kinect 3D data and then used the Chamfer distance [81] to match real postures to 82, 560 synthetic hand shapes. In our approach, we use a slightly modified version of their method for hand–palm separation.

Ren et al.[13] required the user to wear a black bracelet on the gesturing hand's wrist, for easier hand–palm separation, and performed finger detection using a distance-based profile of the palm and shape decomposition. The same authors proposed Finger-Earth Mover's Distance (FEMD) to perform posture recognition. In this work we use the same dataset as in [13], but we avoid the use of a black bracelet, since it might be restrictive in certain cases, such as *"What if the user loses the black bracelet?"*.

In general, posture locking is not explicitly discussed in most works. Implicitly, one may assume that it is based on detecting long time periods where hand positions do not vary significantly.

2.3.3 Recognizing body gestures

Recently, there is a trend for recognition of complicated gestures, which drift from the strict definitions of dynamic gestures or static postures and approach sign language signs or even human activities, such as hand clapping and waving. We refer to these broader types of gestures as *body gestures*, since in many cases hand position and configuration are not sufficient for accurate recognition. Indeed, related literature reveals that a lot of standard human activity recognition methods have been already successfully applied to recognize body gestures. Although our work didn't consider this class of problems, we briefly review some of the most important approaches of the last two years (2012 - 2014) for completeness.

Kaâniche and Brémond [82] learned Local Motion Signatures [83] of Histograms of Oriented Gradients (HOG) [48], combining advantages of global and local gesture motion approaches to improve recognition quality. Their experiments were conducted on mixed datasets, containing both body gestures and generic human activities.

Wu et al.[44] used the Extended Motion-History-Image to represent body gestures and performed one–shot learning on a challenging dataset. Yang et al.[84] discovered high–level motion primitives by hierarchical clustering of optical flow in four-dimensional,

spatial, and motion flow space, achieving high recognition results on both activity and gesture datasets.

Wan et al.[54] proposed Class–Specific Maximization of Mutual Information (CSMMI), a dictionary learning technique for sparse representation–based classification, and conducted experiments on several activity and gesture datasets. Although CSMMI provided high recognition accuracies for all datasets, STIP features [85] performed better for activity datasets, while enhanced motion SIFT features [86] and HOJ3D [87] performed better for gesture datasets.

With the release of Kinect^{TM} , a device that allows automatic tracking of human joints over time, a lot of works used this information for recognition purposes [88]. Hussein et al.[89] described a sequence of skeleton joint locations using multiple fixed length covariance descriptors [90] over sub-sequences in a hierarchical fashion. Final classification was based on Support Vector Machines (SVM) [45], resulting in 93.6% recognition accuracy for 12 body gestures.

Zhao et al.[91] used normalized distances of pairwise joints to describe the skeleton sequence and DTW for online spotting on continuous skeleton streams. Chaaraoui et al.[92] used an evolutionary algorithm to determine the optimal subset of skeleton joints, taking into account the topological structure of the skeleton. Negin et al.[93] discriminatively optimized a random decision forest model to identify the most effective subset of skeleton joints, and then used SVM for recognition on the selected features. Ellis et al.[94] explored the trade–off between recognition accuracy and observational latency, reporting a strong positive correlation between the two quantities. Shen et al.[33] proposed an exemplar–based method that learns to correct the initially estimated human pose, reporting significant improvements on both joint-based skeleton correction and recognition accuracy.

2.4 Discussion

In this chapter, we reviewed the majority of the most important recent gesture recognition approaches found in literature. In most cases, the type of sensor defines the types of gestures that can be recognized (dynamic trajectories, hand postures, complex signs), as well as the feature extraction methods. While most works focus on isolated recognition, a few recent approaches studied online gesture spotting on continuous data streams, proving either direct or indirect solutions. Finally, we reviewed some very recent approaches for recognition of body gestures, which lie in the limits between gesture and human activity recognition.

Part II

Feature extraction

Chapter 3

Data acquisition

Contents

In this chapter, we provide a brief overview of the three data acquisition methods we use, i.e. *body sensors*, *2D cameras* and *depth cameras*. We also describe in detail the datasets we use in our experiments, pointing out their main uses in the various gesture categories (*dynamic trajectories vs static hand postures*, *digits vs letters*). This chapter serves as a reference for subsequent chapters.

3.1 Data acquisition

3.1.1 Body sensors

Body sensors, such as accelerometers in mobile phones or in the Nintendo (R) Wii™ remote control, offer an efficient and easy way for hand detection. Their main advantage is detection accuracy, since the device is typically held by the user's wrist. However, forcing the user to wear or hold a device may limit the naturalness of interaction.

Some of the datasets used in this work were captured by the Nintendo (R) Wii^{TM} remote control, which contains an accelerometer to sense acceleration along three axes. It also contains an optical sensor, to determine the direction of pointing, although we didn't exploit this feature in our experiments.

3.1.2 2D cameras

Standard 2D cameras capture the visible spectrum of light and provide color images. In general, 2D cameras are much cheaper than depth cameras and body sensors, while they are also included in most devices, such as laptops, smartphones and tablet PCs. However, they are very sensitive to light variations, while the lack of depth becomes an important issue when objects occlude each other.

In this work, we use external 2D cameras in two main image resolutions, VGA (640×480) and QVGA (320×240), at a frame rate of $25 - 30$ frames per second (fps).

3.1.3 Depth cameras

Depth cameras offer a disparity image, showing relative distances to the camera. Depth field may be acquired using two standard 2D cameras (*stereo camera*) or special devices which typically operate beyond the visible light spectrum.

In this work, we use Microsoft (R) Kinect^{TM}, which captures both depth and color information. Color image is acquired using a standard 2D VGA camera (640×480, 30 fps), while the depth sensor consists of an infrared radiation projector combined with a monochrome CMOS sensor. VGA resolution is also adopted for depth images, with depth values having a range of $[0, 2047]$ (11–bit depth). Typically, lower values denote distances closer to the camera. The device offers a depth range of $1.2 - 3.5$ m (3.9–11.5 ft) and an angular field of view of 57° horizontally and 43° vertically. Finally, Kinect offers multi-array microphone and voice recognition capabilities, although we don't use this feature in this work.

3.2 Gesture datasets

In this section, we describe in detail the gesture databases we used in our experiments. All datasets are summarized in Table 3.1.

TABLE 3.1: Datasets used in our experiments

	Dataset	Abbreviation	Source
1a	Graffiti Green Digits	GreenDigits	Alon et al.[2]
1b	Graffiti Easy Digits	EasyDigits	
2a	Kinect Train Digits	KinectR	Frolova et al.[10]
2b	Kinect Test Digits	KinectT	
2c	Kinect Non Digits	KinectNonDigits	
2d	Kinect Streamed Input Ordered	KinectStreamOrdered	
2e	Kinect Streamed Input Repeat	KinectStreamRepeat	
3	6D Digits	Digits6D	Chen et al.[3]
4	6D Lower–case Letters	Lower6D	
5	6D Upper–case Letters	Upper6D	
6	6D Digits and upper–case Letters	Alphanumeric	
7	Motion Words	Words6D	
8	KinectPostures	Hand Postures	Ren et al.[13]

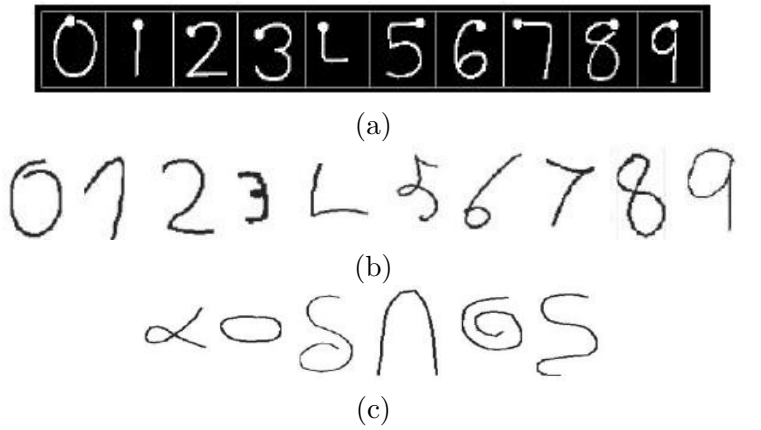
3.2.1 Datasets with continuous trajectory gestures

For our experiments on trajectory recognition, we used 3 publicly available databases for our experiments, namely the 2D Graffiti [2], Kinect [10] and 6DMG [3]. These databases can be split in smaller datasets that test a specific subset of gestures (digits, lower letters and upper letters).

The 10 Palm Graffiti Digits database [2] contains standard QVGA (320×240, 30 fps) 2D videos of 10 users writing *"on the air"* the 10 Hindu-Arabic numerals, $0 - 9$, in a continuous mode [1]. This database is split in three subsets, namely the "GreenDigits", "EasyDigits" and "HardDigits" datasets, out of which we used the first two for our experiments. Each dataset contains 300 gestures in total (10 users $\times$ 10 digits $\times$ 3 examples/digit/user). GreenDigits is used for training, since users wear a green glove, while EasyDigits, in which users wear short–sleeves, is used for testing in a standard user–independent mode. Each video shows the 10 digits in order, i.e. $0, 1, \ldots, 9$.

The third dataset ("HardDigits") is intended for evaluation of advanced hand detection methods under very challenging and uncontrolled scenarios (people moving in the background, trying to confuse the recognition system). However, such complex environments tend to appear quite rarely in realistic indoor systems. On the other hand, videos of "EasyDigits" capture users in a typical office environment, with realistic background (other people working in their desk). Nevertheless, complex backgrounds may often appear in outdoor scenes, and thus addressing such issue is included in our goals for the future.

[1] Available at http://vlm1.uta.edu/~athitsos/projects/digits



(a)

(b)

(c)

FIGURE 3.1: (a) The 10 Palm Graffiti digits, used in [2]. (b) The 10 Palm Graffiti digits, as used in [10]. (c) Out–of–vocabulary gestures, used in [10].

The Kinect database [10] involves the 10 Palm Graffiti digits as well, as captured by a Microsoft (R) Kinect™ device [2]. It consists of rotated (x, y) hand coordinates after applying Principal Component Analysis to the original (x, y, z) data. The database is composed of 5 subsets; a training set, "KinectR", featuring 8 users and 979 isolated gestures in total, and four testing sets – "KinectT", featuring 2 additional users and 122 gestures in total, "KinectNonDigits", containing 50 non-digit gestures (Fig. 3.1-c), "KinectStreamOrdered", containing 7 continuous sequences with 70 gestures in total (the 10 digits in order, $0, 1, \ldots, 9$) and "KinectStreamRepeat" which contains 47 continuous sequences, each showing one digit repeated several times (387 gestures in total).

The 6DMG database [3] contains trajectories written *"on the air"* using a Nintendo (R) Wii™ device, at 60 fps [3]. This database is split in many subsets out of which we use those three that contain letters and digits, namely the "Digits6D" (6 users, 600 gestures in total), "Lower6D" (5 users, 1300 gestures) and "Upper6D" (13 users, 6500 gestures) subsets. The last 2 subsets correspond to the lower– and upper–case letters of the English alphabet, a–z and A–Z, correspondingly. Although the Wii device provides additional features (such as depth and speed), we keep only the (x, y) coordinates corresponding to the device position, to allow for a fair comparison among all datasets. To our knowledge, this is the first work that uses these specific subsets, since prior work concentrated on individual strokes [42, 95].

We also use the "Motion Words" subset, which contains continuous streams of 40 words, written on the air by 12 users (40 words, 5 examples/user/word, 2400 sequences in total). These sequences are split in 4 sets, each containing 10 words with an average of 4 characters per word, corresponding to 4 different scenarios of using gestures to control

[2]Kindly provided by Professor Sigal Berman and Dr. Darya Frolova, with the Department of Industrial Engineering Management, Ben-–Gurion University of the Negev, Beer–Sheva, Israel.

[3]Available at http://www.ece.gatech.edu/6DMG/Download.html

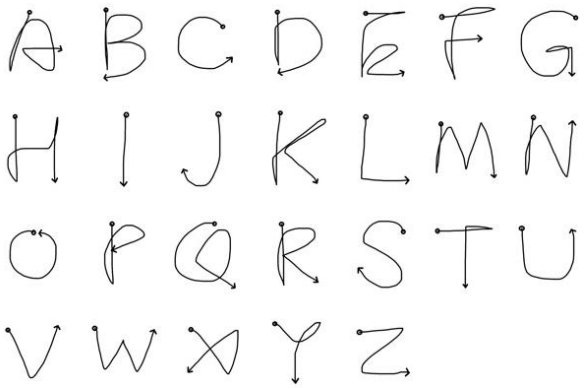


FIGURE 3.2: The 26 upper–case English letters, used in [3].

an integrated entertainment system, such as selecting TV channels, music genres and basic internet websites, as shown in Table 3.2.

TABLE 3.2: The 4 scenarios included in the *Motion Words* dataset

Set 1	**Set 2**	**Set 3**	**Set 4**
ABC	BBC	WEATHER	GAME
CBS	FX	NEWS	VOICE
CNN	HULU	MLB	CALL
DISCOVERY	TNT	NFL	MAIL
DISNEY	MUSIC	TRAVEL	MSG
ESPN	JAZZ	POKER	FB
FOX	ROCK	FOOD	YOU
HBO	DRAMA	KID	GOOGLE
NBC	MOVIE	MAP	SKYPE
TBS	SPORT	TV	QUIZ

3.2.2 Datasets with postures

For our experiments on hand posture recognition, we used the "KinectPostures" dataset [13], which contains an alphabet of 10 different postures, with 10 examples from 10 different persons, i.e. 1000 postures in total (Fig. 3.3). In their original work, Ren et al.[13] required the user to wear a black bracelet on the gesturing hand's wrist, for easier hand–palm separation. In our work, we discard the presence of a black bracelet, since it might be restrictive in certain cases, such as *"What if the user loses the black bracelet?"*. Thus, we restrict the use of pixel luminance values to detect face location during initialization, and we only use depth information for all the other tasks, i.e. hand detection, segmentation and representation.

FIGURE 3.3: The 10 different hand postures used in [13] and in our experiments. One can observe the necessity of using depth under highly correlated background (e.g. in (b,c)).

3.3 Discussion

This chapter serves as a reference for subsequent chapters, providing a brief overview of the data acquisition methods (*body sensors*, *2D cameras* and *depth cameras*) and datasets we use in our experiments.

Chapter 4

Feature extraction

Contents

Feature extraction is typically the first step in gesture recognition systems, involving hand detection and gesture representation. In this chapter, we first present our approach for hand and palm detection under various data acquisition methods, and propose a novel technique for finger detection. Finally, we describe our approach for gesture representation, targeting invariance to translation, rotation, scaling and nature of the capturing device.

4.1 Hand detection

Gesture capturing devices affect the choice of hand detection method, posing various limitations on the types of possible gestures and suitable signal processing tools (Sec.

2.1). In the following we discuss hand detection methods we used for the cases of body sensors, depth cameras and standard 2D cameras.

4.1.1 Hand detection using body sensors

Hand direction can be achieved directly when body sensors are used. Specifically, a device is attached to the gesturing hand, making coordinates easily available for further processing. In the cases of mobile phones or the Nintendo (R) Wii ^{TM} remote control, the user simply holds the device while gesturing. Clearly, in such case one implicitly assumes that the user is capable of holding an object steadily in his hand, which requires a certain anatomical and functional capacity.

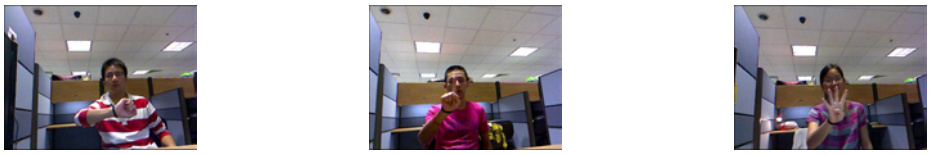
4.1.2 Hand detection using depth cameras

In the case of depth cameras, such as Kinect, hand detection is usually based on the assumption that the gesturing hand is the closest object to the camera and thus simple or advanced depth thresholding techniques and tracking methods can be employed with very accurate results [72], [15], [13]. In the following, we present our hand detection method for the case of depth camera sensors in detail.

4.1.2.1 Face detection

The first step in our approach performs face detection on the luminance (Y) component of a color image, based on the Viola–Jones AdaBoost cascade classifier [96]. This technique partitions an image into square blocks (at various scales) and then assigns a classification label (*face/no face*) to each of them. The main idea is to discard many negative blocks at early stages of the cascade, while assigning a positive label (*face*) to blocks reaching the final cascade level. For our experiments, we used an efficient implementation, provided by the OpenCV library [20].

Presence of a face indicates existence of a user and triggers the system for further processing. We inherently assume that face detection will be successful at a certain time instance, meaning that the user will not cover his face with his hand when he first appears on the screen. Based on face position and user's laterality (e.g. right–handedness), we can exclude the opposite image plane (e.g. left of the face) from further analysis. Laterality can be easily inferred by observing the location of hand motion relative to the user's head during some first initialization frames. Finally, depending on the application,



(a) (b) (c)

FIGURE 4.1: Three of the 10 different hand postures used in [13] and in our experiments. One can observe the necessity of using depth under highly correlated background (e.g. in (b,c)). While the users wear a black belt in their wrist for easier hand detection and separation, we completely avoid using such information, by processing only the depth image.

we can also eliminate some part of the image plane over the face since gesturing rarely occurs there.

When a user is eventually detected, we compute the average depth of the face region, T_f, which is the maximum depth at which the gesturing hand can occur, assuming it is always closer to the camera compared to the distance of the face, as shown in Fig. 4.1.

While the cascade classifier provides increased efficiency compared to more naive implementations, face detection still remains a computationally heavy process, even for modern high–end personal computers. Since a person normally does not move his head wildly during gesturing, face detection can be applied periodically every F_t seconds, using the most recent face location in some short history for subsequent frames. In our implementation, F_t varies, being smaller ($F_t = 1$ second) when there is no prior positive face detection result, and larger ($F_t = 5$ seconds) when a user has already been located on the screen. This technique is computationally efficient, while it also deals with hand–face occlusions, since face detection will eventually succeed even after some unsuccessful attempts.

4.1.2.2 Arm detection

This step takes as input a depth image and the average depth of the face region, T_f, and outputs a binary mask indicating the points of the arm component, as in Fig. 4.2–c.

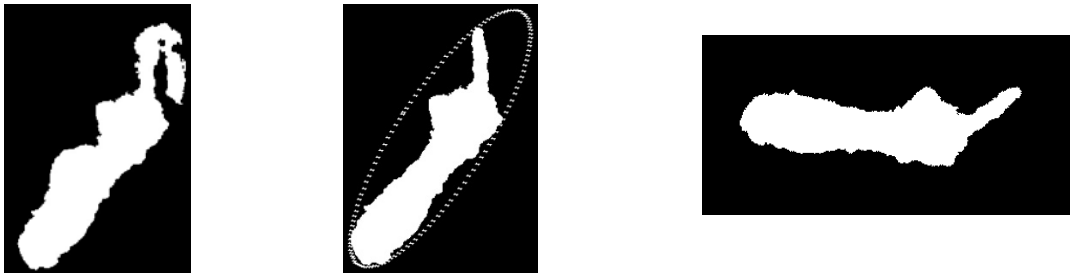
We first apply depth thresholding, keeping pixels with depth $d < T_f - T_0$, where T_0 is a small value that typically represents the minimum distance from the face plane to the waving hand – we found out that a value of $T_0 = 100$ is suitable for our dataset. Subsequently, we perform Connected Component Analysis (CCA) and keep the biggest component as the candidate arm (Fig. 4.2–a).

Although depth thresholding helps dealing efficiently with cluttered background, certain problems may arise due to periodic applications of face detection (Sec. 4.1.2.1).

Specifically, while T_f is considered constant for consecutive frames, it usually presents slight variations, as the user moves his face unconsciously. This leads to less accurate segmentation, causing problems when the hand is in front of the face, as the thresholded mask will contain both the hand and some part of the face (Fig. 4.2–a).

This problem can be solved efficiently by inspecting the histogram of depth values (Fig. 4.3) and applying Otsu's segmentation algorithm [97] to separate it into two components. Otsu's algorithm also returns a metric of *separation quality* to protect against cases where there exists only one component. As it can be seen in Fig. 4.2-b, the final result is the desirable one. Our literature review revealed that, in a similar approach, Kurakin et al.[72] also used Otsu's algorithm to segment the arm from the whole body.

Finally, similar to [14], we compute the Minimum Enclosing Ellipsoid (MEE) [98, 99] to find the orientation of the arm (through the axis of elongation) and rotate the arm in a horizontal position, such that the palm is always at the right side, as shown on Fig. 4.2-c. Such rotation is achievable, based on the assumption that the palm area will have a lower depth value compared to the rest of the arm, as shown in Fig. 4.4. We call the result of this transformation *standard position* 0°.



(a) (b) (c)

FIGURE 4.2: (a) Resulting binary mask after depth thresholding. (b) Resulting binary mask after applying Otsu's segmentation. One can see that the shape of the hand is now cleaned from background noise. Minimum Enclosing Ellipsoid indicates arm's orientation. (c) Arm rotated at the *standard position* 0°.

4.1.2.3 Hand – palm separation

This step takes as input an arm component in the *standard position* 0° (Fig. 4.2-c) and outputs a cutting band at the wrist (Fig. 4.5-e).

In the following, let's assume Cartesian coordinate system, with the $(0, 0)$ point in the lower–left corner of each image. For hand–palm separation, restricting ourselves in the bounding box of the arm, we scan all x-positions of pixels that belong to the largest component and track the pixels that have the minimum ($low(x)$) and maximum

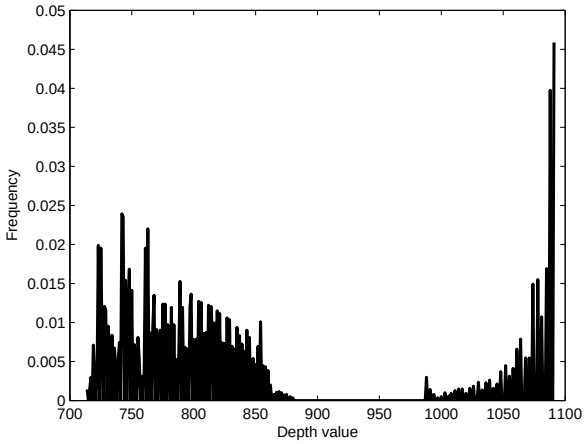


FIGURE 4.3: Histogram of depth values for the arm mask (Fig. 4.2-a). Note arm pixels occur at small depth values, while the noisy part appears at bigger depth values.

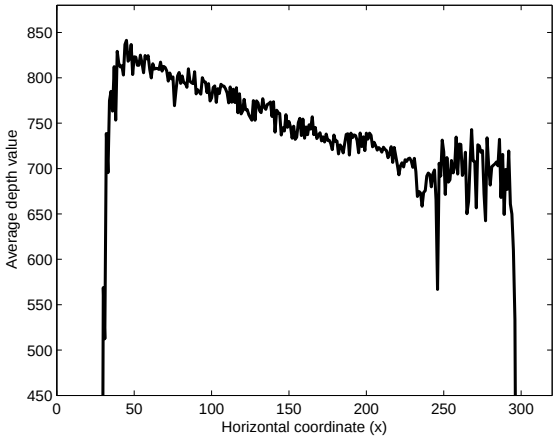


FIGURE 4.4: Average depth profile of the arm shown in Fig. 4.2-c. Note that fingers lie at lower depth values, i.e. closer to the camera, as expected.

($upp(x)$) y-coordinates. These two sequences form two 1D-signals, as shown in Fig. 4.5-b. Subtracting these two signals produces the signal of widths $widths(x)$ (Fig. 4.5-d).

We observe that the arm–palm separating point, (x^*), can be located at the local minimum of $widths(x)$ that is closest to the global maximum, x_M. At this point, the reasons for rotation to the *standard position* 0° become clear: searching for a local minimum is restricted only to the left part of the signal, i.e. to positions $x < x_M$. This way, we avoid erroneous segmentations due to local minima after the true wrist point. As an example, in Fig. 4.5-d, $x_M = 266$, $x^* = 210$, while a false local minimum lies at $x = 311$. The final separation result is a mask M_0 containing the palm, as shown in Fig. 4.5-e.

We note that our method is slightly different from [14], which performed some smoothing on the 2D contour, derived a signal similar to $widths(n)$, choosing as starting point the global maximum and applied gradient descent towards the end of the hand until the

algorithm reached a local minimum. While in our experiments we saw that the method of [14] works fairly well in practice, we still found cases where it fails, because of not applying enough smoothing or because the global maximum does not always reside at the palm area.

Hand–palm separation was visually confirmed, with results of similar quality to the example shown in Fig. 4.5-d. The only exception was one case (out of 1000) where the resulting mask missed the thumb and placed the hand–palm separation line incorrectly, in the middle of the palm.

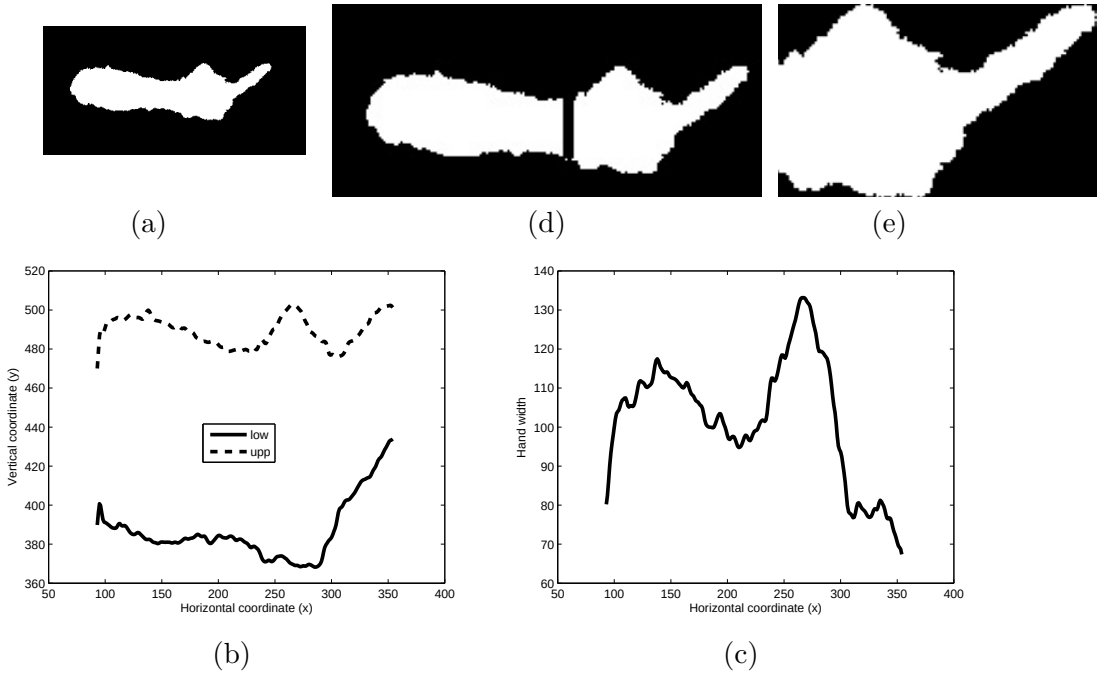


FIGURE 4.5: (a): *Standard position* 0° for the hand of Fig. 4.2. (b) The corresponding signals $upp(n)$ and $low(n)$. (c) The signal of widths, $widths(n)$. (d) The detected cutting point, shown with a vertical red line. (e) The detected palm mask.

4.1.3 Hand detection using 2D cameras

Hand detection is more complicated when only one 2D camera is used. Difficulties are mainly due to the projection of 3D objects on a 2D image plane. In this case, object segmentation depends exclusively on color information and motion. Some early approaches asked the user to wear a special glove, colored to some distinct color which rarely appears in most background scenarios (e.g. green). Typically, simple green color detection leaves only one object present on the image plane, denoting palm of the moving hand. As this technique limits the naturalness of interaction, it is used only during training, to ensure recording of clean training data. In this work, we applied green color detection on the "GreenDigits" dataset (Table 3.1).

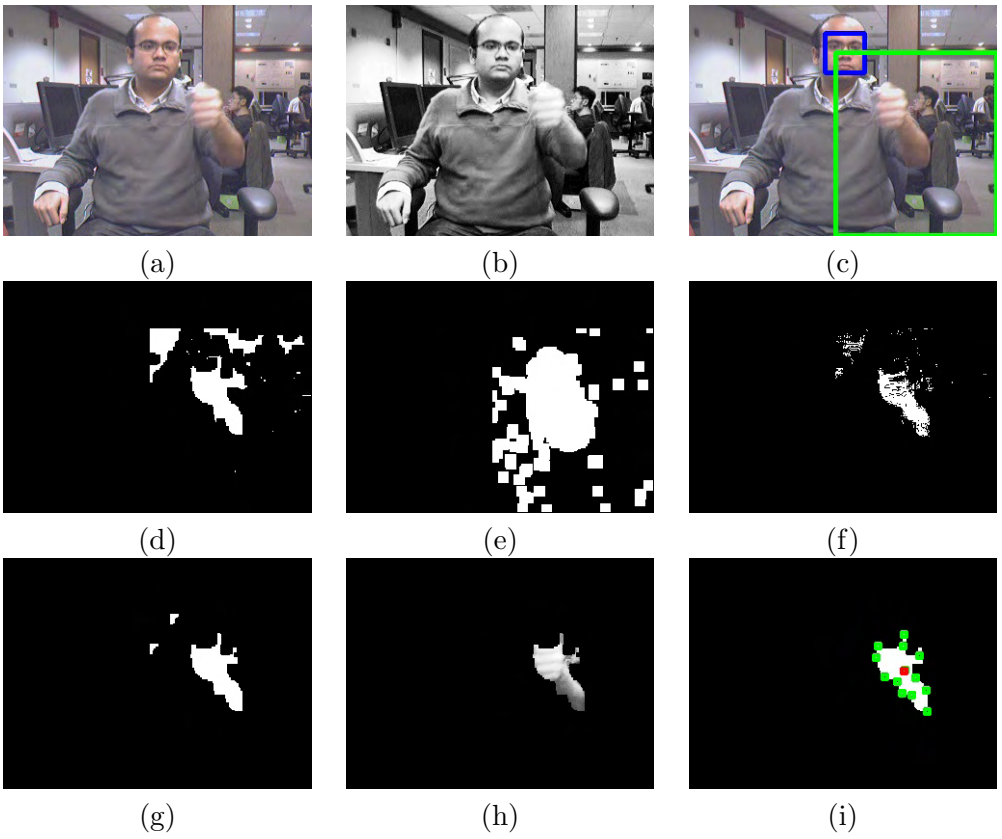


FIGURE 4.6: (a) A characteristic RGB frame from the Graffiti "EasyDigits" dataset [2]. (b) Corresponding luminance (Y) component. (c) Result of face detection (*blue box*) and the corresponding area where hand detection will be applied (*green box*). User's laterality is inferred during initialization and is considered known at this step. (d) Skin mask after skin detection and thresholding. (e) Motion mask after frame differencing of consecutive frames. (f) Combined skin–motion mask, using the logical AND operation. (g) Skin–motion mask after applying morphological opening for noise removal and hole filling. (h) Resulting hand region. Green dots represent the interesting points found through corner detection, while the red dot in the center represents the estimated palm centroid.

During testing, the user gestures using bare hands. In such cases, skin–color detection is a fundamental part of most hand detection systems, combined with some form of motion detection. In our work, we developed a hand detection method, based on face detection, skin color detection and corner detection techniques, as follows.

4.1.3.1 Face detection

As in the case of depth cameras (Sec. 4.1.2.1), face detection is performed on the luminance (Y) component of a color image, based on the Viola–Jones AdaBoost cascade classifier [96] and the OpenCV implementation [20]. Once again, presence of a face indicates existence of a user and triggers hand detection in a restricted area, based on user's laterality and face location (Fig. 4.6-c).

4.1.3.2 Skin detection

Skin color detection is performed using the generic color histogram method of Jones and Rehg [100], which computes a skin probability image. The skin probability of a pixel with color (r, g, b) is computed using Bayes theorem:

$$P(skin|rgb) = \frac{P(rgb|skin) \cdot P(skin)}{P(rgb)} \tag{4.1}$$

while the non–skin probability is:

$$P(\neg skin|rgb) = \frac{P(rgb|\neg skin) \cdot P(\neg skin)}{P(rgb)} \tag{4.2}$$

where $P(rgb|skin), P(rgb|\neg skin)$ denote the conditional probabilities of observing the triplet (r, g, b) in skin and non–skin images, while $P(skin), P(\neg skin)$ denote some prior probabilities. In their original work [100], Jones and Rehg estimated these quantities using 80 million skin pixels and 854 million non–skin pixels from natural images. Classification is performed by comparing:

$$P(skin|rgb) \geq \theta P(\neg skin|rgb) \tag{4.3a}$$

$$P(rgb|skin) \cdot P(skin) \geq \theta P(rgb|\neg skin) \cdot P(\neg skin) \tag{4.3b}$$

$$\frac{P(rgb|skin)}{P(rgb|\neg skin)} \geq \theta \frac{P(\neg skin)}{P(skin)} \tag{4.3c}$$

$$\frac{P(rgb|skin)}{P(rgb|\neg skin)} \geq \Theta \tag{4.3d}$$

where θ is a parameter related to the application–dependent cost of false positives and false negatives. In general, one can consider only parameter Θ, which controls the the trade–off between true and false positives. In their original work, Jones and Rehg reported high quality classification results on their dataset (*Equal Error Rate* – $EER = 0.88$). Finally, to reduce memory requirements, they also used 32^3 bins for each histogram (instead of 256^3 = total number of 8–bit RGB colors).

For our experiments on the "EasyDigits" dataset, we used the 32–binned histograms with $\Theta = 0.45$. We also precalculated the final classification results for all bins, reducing the required memory space to 32^3 bytes instead of 2 *histograms* $\times 32^3$ *bins* $\times 2$ *bytes per floating–point number*. In our experiments, we observed an average computational speedup of $3\times$. However, this implementation returns directly a binary skin mask instead of a skin probability mask, which may be undesirable in some applications.

Fig. 4.6-d shows the resulting skin mask, M_{skin}, after applying skin detection on the frame of Fig. 4.6-a. One can easily notice the appearance of true positives (user's face and arm) and false positives (skin–like background due to other humans, lighting and wooden furniture).

4.1.3.3 Motion detection

The resulting skin mask, M_{skin}, can be combined with some binary motion mask M_{motion} (Fig. 4.6-e), producing a skin–motion mask, $M_{skin-motion}$ (Fig. 4.6-f). Typically, M_{motion} results after applying a threshold Th_{Motion} on the luminance difference image of consecutive frames. Determining Th_{Motion} depends on environment conditions (varying lighting, dynamic background) and can be estimated assuming that the user will not appear in the first few frames.

For our experiments, we used the fixed value $Th_{Motion} = 15$, which gave conservative results, as shown in Fig. 4.6–e. While our principal goal was to keep as many true positives as possible, we observe that major false positive regions, i.e. face and background, are almost completely eliminated due to low amount of motion in those regions. In fact, this is a common condition in most office environments.

The resulting mask can be further improved through morphological operations, which typically remove small noisy objects, while they also fill holes in bigger objects (Fig. 4.6–g). Once again, the exact number and order of such operations is application–dependent and there is no standard procedure to decide an optimal strategy. In our experiments we applied one opening operation, followed by two dilation operations.

4.1.3.4 Final hand detection

The previous steps typically remove most of the noise, while leaving one large object on the foreground (Fig. 4.6-g). At this point, we perform Connected Component Analysis (CCA) [20] and isolate the arm as the largest component within the luminance frame (Fig. 4.6-h).

Unfortunately, we cannot apply our method developed in Sec. 4.1.2.3 for arm–palm separation, as the resulting mask is too noisy, due to reduced resolution and imperfect thresholding operations in the previous steps. For this reason, we apply strong corner detection, using the method of Harris and Stephens [101] to collect points of interest. We then calculate their optical flow using the pyramid–based Lucas–Kanade method [102] and discard all points showing motion vectors of low magnitude – we use a threshold

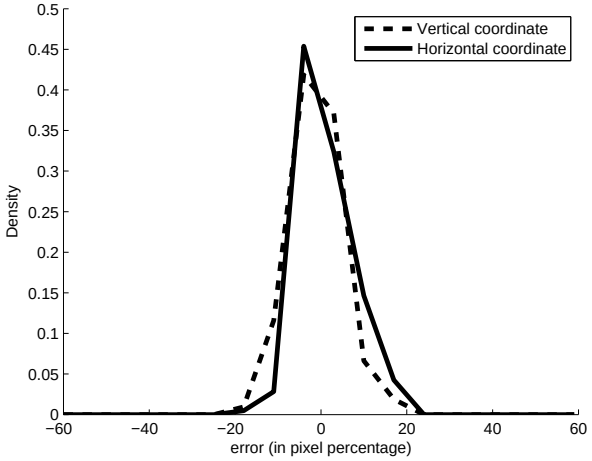


FIGURE 4.7: Histograms of error in x and y palm coordinates, expressed in % *(pixel offset/pixel frame size)*, for 423 test frames. Fit a Gaussian with mean $\mu = 0.9\%/-1\%$ *(horizontal/vertical errors)* and standard deviation $\sigma = 6\%$.

$T_{flow} = 0.5\%$ of the frame width. Finally, we choose the centroid of the remaining corner points as the final palm centroid (Fig. 4.6-i).

We can expect this method to find a point closer to the palm center than the rest of the arm, since palm and fingers present most of the edges and corners. Although optical flow estimation is computationally expensive, that step is required in order to avoid confusions with static skin–like image parts, such as face, when the hand is not moving. While we observed very small differences in our experiments on the Graffiti EasyDigits dataset (Table 3.1), optical flow estimation proved quite helpful when we conducted additional experiments with a real–time application. Although the steps described earlier are not new, to our knowledge, using corner points for hand detection is novel.

Please note that our method would not work properly in scenes with very complex backgrounds, i.e. too many skin–colored moving objects [2]. However, such scenarios are rather unrealistic in typical indoor environments (home/office).

4.1.3.5 Evaluating hand detection

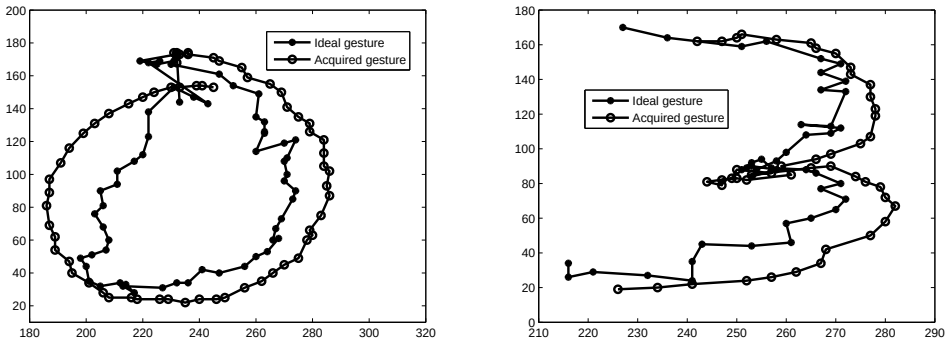
To estimate the noise introduced by our hand detection method, we manually labelled the *optimal* hand position in 423 frames (VGA, 320×240) from the EasyDigits dataset (Table 3.1). These frames contain the digits "0" – "5" and the connecting lines between consecutive digits.

Fig. 4.7 shows the estimated probability density function (*pdf*) of the noise for horizontal and vertical normalized coordinates (depicted as 100% percentages of the image

dimensions). It appears that noise can be modelled as additive Gaussian of zero mean. A standard measure of noise is the Signal–to–Noise–Ratio (SNR), defined as:

$$SNR = 10 \log \frac{\sigma^2_{signal}}{\sigma^2_{noise}} \tag{4.4}$$

In our data, the estimated $SNRs$ are 21.5 dB for horizontal and 21 dB for vertical coordinates. These values are generally considered as moderate–to–low. However, the shapes of the gestures seem not be severely distorted and remain distinguishable by the algorithms presented in Chapter 5 (Fig. 4.8).



(a) (b)

FIGURE 4.8: Ideal and acquired trajectories for digits (a) "0" and (b)"3". Despite the low SNR (21.2 dB), the basic shape features of the digits remain distinguishable and can be recognized by the gesture recognition methods presented in Chapter 5.

4.2 Finger detection

Location, formation and number of fingers offer valuable information during recognition of hand postures. In this section we propose an automated method for reliable finger detection, operating on a binary mask of hand's palm. We note here that our method does not rely on color or depth information, and thus it can be potentially used with any type of video camera, provided that hand segmentation is successful. In our experiments, we observed excellent performance on the "KinectPostures" dataset (Table 3.1), which was used for posture recognition.

In the following, we describe our approach in detail, while in Chapter 9 we show that recognition accuracy of standard hand–shape representation methods can be significantly improved, through search space reduction based on the number of fingers.

4.2.1 Initialization

Our method assumes that hand detection and hand–palm separation have been successfully performed, resulting in a binary palm mask, as shown in Fig. 4.9-a. At first, we estimate the palm's radius, R_p, by solving for the Maximum Inscribed Circle [103] (Fig. 4.9-b). We then apply morphological image opening on the mask M_0 with a disk of radius $R_{disk} = 0.5 \cdot R_p$, in order to keep only the palm M_{palm} (Fig. 4.9-c). Subtracting M_{palm} from M_0 keeps mainly the fingers (Fig. 4.9-d). Finally, we remove small, noise-like components in the mask by additional morphological opening with a smaller structuring element, which results in the mask $M_{fingers}$ (Fig. 4.9-e).

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

FIGURE 4.9: (a) Binary hand mask, M_0, resulting after hand palm separation in previous steps. (b) Corresponding maximum inscribed circle. (c) Estimated palm M_{palm}, after morphological opening. (d) Subtraction of the two masks $M_0 - M_{palm}$. (e) Resulting mask $M_{fingers}$ with candidate finger components.

4.2.2 Dealing with merged fingers

While the above procedure is quite simple and fast, two problems may appear:

1. *False Fingers.* Some components may survive the preprocessing step because they are large enough and not because they have a valid finger shape.
2. *Merged Fingers.* Some fingers may appear as one component due to viewpoint or to device artifacts.

To alleviate these problems, we propose a novel method for finger detection, which first detects candidate fingers and then uses the mask $M_{fingers}$ to confirm the validity of the results. Similar to [13], we find the contour coordinates $\{x(n), y(n)\}$ of M_0 and then form the signal of radial distances with respect to the palm center, $r(n)$:

$$r(n) = ((x(n) - c_x)^2 + (y(n) - y_c)^2)/R_p \tag{4.5}$$

where (c_x, c_y) denotes the palm centroid coordinates of M_{palm}.

As we see in Fig. 4.10, the signal $r(n)$ presents apex–shaped lobes around finger areas. This property was also observed by Ren et al.[13], who used near-convex hand decomposition [104], obtaining accurate finger detection results, but at a high computational cost. In our work, we propose a much simpler approach, based on a motion analysis method [105] and a novel method for apex detection.

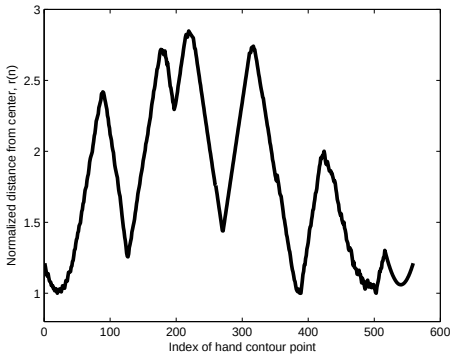


FIGURE 4.10: The signal of radial distances, $r(n)$, for the hand shown in Fig. 4.9-e. One can easily note the apex–shaped lobes around finger areas.

4.2.2.1 Detection of Full Action Instances

Full Action Instances (*FAI*) correspond to periodically appearing cycles in human activities [105], e.g. a step in the walking activity. Poularakis et al.[105], noticed that FAIs correspond to apex-shaped parts in a properly–constructed 1D signal (*average motion energy*), and proposed a method for FAI detection. In short, our algorithm forms teams of two minima and one maximum and tries to combine neighbouring teams, using 16 combining rules (Table 3 in [105], Fig. 4.11 below); each FAI is roughly a mountain peak between two valleys.

FIGURE 4.11: Three (out of sixteen) rules of combining neighbouring teams of two minima and one maximum. In cases (a,b), two smaller teams are combined to make a larger team. In case (c), no combination is available, which results in a FAI detection for the previous complete bell–shaped object.

In our approach, we use this algorithm for finger detection, since fingers resemble FAIs. After the initial FAI detection (Fig. 4.12), we reject small FAIs based on a minimum apex height, T_h, to obtain the final candidate fingers. This process removes false fingers

from the mask $M_{fingers}$, since it keeps only apex-shaped components of the hand. In our experiments, we used $T_h = 0.3 \cdot R_p$, i.e. we require a finger's length to be at least a third the radius of the palm

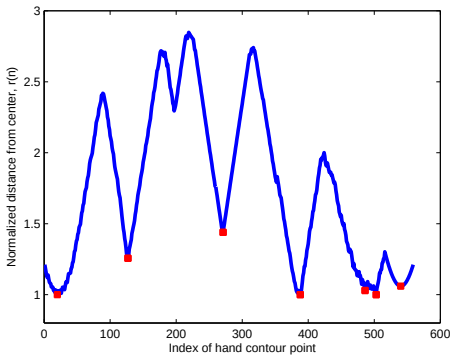


FIGURE 4.12: The signal of radial distances, $r(n)$ and the detected FAIs (*shown as red squares*), using the method of [105]. Note that low-height apexes can be easily rejected.

4.2.2.2 Apex detection

While FAI detection locates the bell–shaped objects quite successfully, it still results in merged fingers, as its original goal was to produce bells as large as possible. However, we observe that merged fingers correspond to significant apexes of the signal $r(n)$. For this reason, we propose a novel algorithm that detects apexes in 1D signals, which we then apply on each FAI.

Our main observation is that an apex can be approximated by a triangle, with area E, as shown in Fig 4.13. Our algorithm begins with the left valley point of the FAI and scans local minima, computing E, the area between the signal (blue curve) and the line connecting the left valley point to the local minimum under test (red line). Depending on the relative position of the two curves, some areas have positive sign (signal lies above the line) while others have a negative sign (signal lies below the line). When $E > T_a$, a significant apex is detected at the point of maximum height. When $E < 0$, we understand that the signal forms a *valley*, which signals our algorithm to backtrack to the last local minimum and restart the whole process from there. Additionally, false apexes can be rejected based on a minimum length of the two sides around the apex, as well as a maximum ratio of the longest over the shortest side. In this way, although originally we assumed one apex per FAI, we can now detect additional apexes and thus disambiguate merged fingers. In our experiments, we normalized $r(n)$ to 1 and then used $T_a = 0.1$.

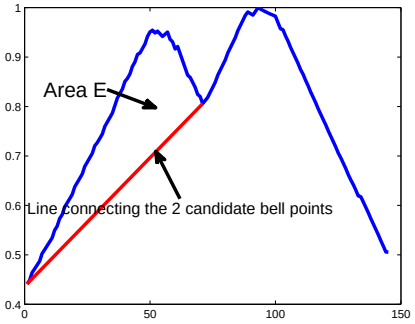


FIGURE 4.13: An example of our method for apex detection and FAI splitting, dealing with two merged fingers. When area E is larger than a threshold T_a, an apex is detected in signal $r(n)$.

4.2.3 Finger verification

At the end of the above process, almost all of the false positives have been eliminated. Accuracy can be further improved, by using the connected components in the mask $M_{fingers}$ for final verification. Since each FAI, with its corresponding apex, left and right sides and valleys, should be restricted to a *single* mask component, we can choose the midpoints at each side of the apex and connect them with a line. If the apex corresponds to a real finger, then most of the points on the line (ideally all of them) will belong to the same component, without intersecting background pixels or other components.

4.2.4 Preliminary evaluation

While a thorough evaluation of finger detection would check spatial matching and boundaries between neighboring fingers, we didn't have access to any reference information for such a task. Moreover, our work differs from Computer Graphics approaches, where the goal is to recover an accurate hand model. Instead, we target posture recognition, using basic geometric features for representation and the number of fingers for search space reduction. Based on the above, we considered as correct cases those where the expected number of fingers - known a priori by the posture class - was returned, with 996 correct results out of 1000 postures.

4.3 Gesture representation

Hand detection extracts the hand region or hand coordinates at each time instance $t = 1, \ldots, T$. The goal of representation is to describe the gesture data in a compact way,

ensuring existence of desirable properties, such as invariance to translation, rotation, time duration and scaling. In the following, we discuss our proposed representation schemes for hand trajectories and postures.

4.3.1 Representing hand trajectories

Hand detection extracts the (x, y) hand coordinates at each frame $t = 1, \ldots, T$, resulting in two 1D signals, $\vec{x} = [x_1, \ldots, x_T]$ and $\vec{y} = [y_1, \ldots, y_T]$. To achieve translation invariance, we subtract the mean values $(\bar{x}, \bar{y})$ from the original observations and form the signal vectors $\vec{x}_{inv} = [x_1 - \bar{x}, \ldots, x_T - \bar{x}]$ and $\vec{y}_{inv} = [y_1 - \bar{y}, \ldots, y_T - \bar{y}]$. Time duration invariance is achieved by downsampling in time through linear interpolation of $(\vec{x}_{inv}, \vec{y}_{inv})$ to obtain two signal vectors $(\vec{x}_r, \vec{y}_r)$ of fixed length N, where N is chosen as a parameter – we used $N = 8$ in our experiments. Finally, we form a single 1D signal $\vec{v}$ of length $2N$ by interleaving $(\vec{x}_r, \vec{y}_r)$, such that

$$\vec{v} = [\vec{x}_{r1}, \vec{y}_{r1}, \ldots, \vec{x}_{rN}, \vec{y}_{rN}] \tag{4.6}$$

and normalize $\vec{v}$ such that its L_2-norm equals 1, in order to ensure scale invariance (i.e. camera pixel resolution, relative distance of gesturing hand to the camera, amount of motion in gesture formation). Fig. 4.14 shows the corresponding representations for digits 0–9 in the "GreenDigits" dataset.

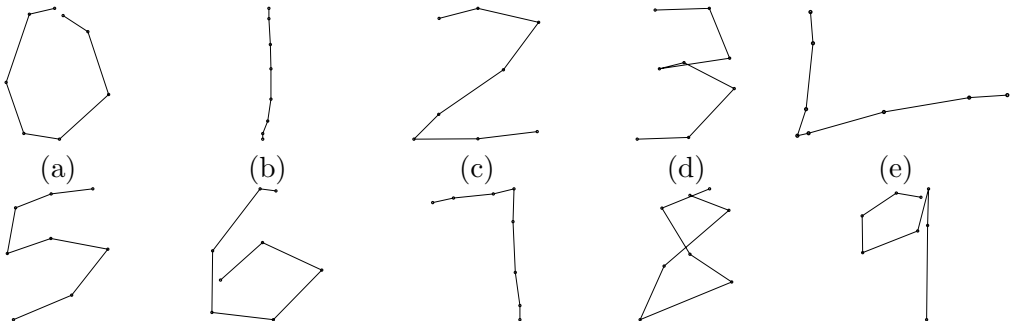


FIGURE 4.14: Digits 0–9, using our 8–point representation. Although fine shape details are not present, digit recognition is still possible.

The reason why we use one interleaved signal vector, $\vec{v}$, instead of two separate vectors, $\vec{x}, \vec{y}$, is due to scale normalization and can be seen in Fig. 4.15. In some trajectories, such as the digits "1, 7" and the letters "I, L", signal values in one dimension – typically the vertical axis – may spread over a wide range of values, while the other dimension presents almost zero variance. In such cases, separate normalization of $\vec{x}$ and $\vec{y}$ may lead to severe shape distortions, which increases the amount of error and may result in wrong classification. As we observe in Fig. 4.15, small distortions in the original trajectory of "1" can have a serious (c–d, separate signal normalization) vs. negligible

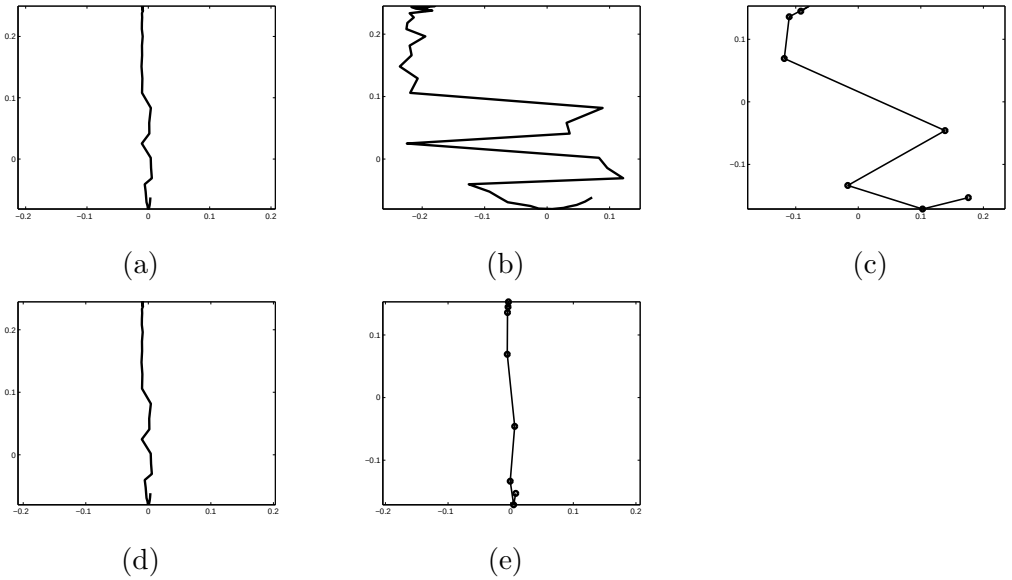


FIGURE 4.15: (a) Original x, y coordinates for digit "1" from Digits6D dataset. (b) Raw coordinates x, y after separate normalization of x and y. (c) 8–point representation of (b) after resampling, under separate normalization scenario. (d) Raw coordinates x, y after common normalization of x and y. (e) 8–point representation of (d) after resampling, under common normalization scenario.

(e–f, interleaved signal normalization) effect on the final representation, depending on the type of normalization we choose. We also observe that downsampling works as a low–pass filter and can thus reduce much of the noise in high frequencies.

While existing works [8] also use downsampling in time, others [6, 7] perform resampling in the arc–length domain, to secure invariance to different speeds during gesturing. In this work we use linear resampling in time since it is computationally simple and performed very well in our experiments.

4.3.2 Representing hand postures

We consider two fundamentally different methods to represent hand postures, a local method and a global one, as described below.

4.3.2.1 Local representation

Hand detection results in a mask, M_0, showing the palm of the gesturing hand. We find the palm contour coordinates, $\{x(n), y(n)\}, n = 1, \ldots, M$ and resample each signal, keeping N values –we chose $N = 256$. We then form the complex contour signal:

$$z(n) = x(n) + jy(n) \tag{4.7}$$

and compute its Discrete Fourier Transform *(DFT)*:

$$F_k = \sum_{n=0}^{N-1} z(n) \mathrm{e}^{-j2\pi kn/N} \tag{4.8}$$

Subsequently, we keep only the $2P$ coefficients corresponding to $k \in [-P, P] - \{0\}$. Finally, we keep only the magnitudes $|F_k|$, normalized by $|F_0|$.

This representation, known as Fourier Descriptors (FD), is invariant to translation, rotation, scaling and choice of initial boundary point [47]. Instead of the contour signal, $z(n)$, Kulshreshth et al.[17] used the signal of distances, $r(n)$, (Eq. 4.5). Its main advantage is that it is a real signal, resulting in symmetric DFT and thus half the number of coefficients. For completeness, we considered and evaluated both approaches in our experiments, using $P = 8$. We refer to these Fourier Descriptor–based representations as $FD(z)$ and $FD(r)$. Please note that resampling to N points is important in both cases, since it guarantees existence (and comparison) of the same frequencies in the DFT representation.

It is also possible to use information about fingers for search space reduction, i.e. one can consider as candidate categories only those postures with the detected number of fingers. We named these approaches $FD^*(z)$ and $FD^*(r)$, corresponding to the previously mentioned $FD(z)$ and $FD(r)$.

4.3.2.2 Global representation

Global representations use *holistic* features, mainly based on the fingers. In our work, we use such information, as derived from our finger detection method (Sec. 4.2). Specifically, we describe a finger using its size characteristics (height, width) and its relative distance from the leftmost finger. Finally, we concatenate all this information for multiple fingers into one feature vector. We refer to this finger–characteristics–only representation as *"Fingers"*.

4.4 Discussion

In this chapter, we presented our hand detection method and a novel technique for finger segmentation under various data acquisition methods. Moreover, we presented our approach for gesture representation, targeting invariance to translation, rotation, scaling and nature of the capturing device. This chapter serves as a fundamental reference for the following chapters.

Part III

Gesture recognition

Chapter 5

Isolated gesture recognition

Contents

Isolated recognition assumes known starting and ending time frames of gestures. In the case of dynamic trajectories, this implies that a sequence of hand coordinates is given as input to the recognition system (Sec. 4.3.1).

In this chapter, we discuss our approach for isolated gesture recognition, which is based on Maximum Cosine Similarity (MCS) and a tree–based fast Nearest Neighbor algorithm [106]. We furthermore discuss two other pattern recognition methods, based on

Dynamic Time Warping–DTW [107] and Support Vector Machines–SVM [45], which are commonly used for gesture recognition [29, 53, 89, 91, 108].

Using MCS for gesture recognition is not novel, as it is already known from the literature [42, 67] that MCS can achieve very high recognition results for stroke–based gestures. Thus, our contribution lies in investigating its usefulness for recognition of digits and letters, exploring various parameters, such as recognition with a varying number of training users/examples, performance on noisy data and common recognition of digits and letters. To this end, we perform extensive results on three publicly available databases (Sec. 3.2). On the other hand, exploiting the computational efficiency of fast Nearest Neighbor techniques for gesture recognition is quite novel and has not been explored in the related literature.

Overall, isolated gesture recognition forms the basis for the two other fundamental operations, i.e. gesture verification (Chapter 6) and continuous gesture spotting (Chapter 7). Hopefully, the results presented in this chapter provide more insight to the details of the other two problems.

5.1 Gesture recognition

5.1.1 Maximum Cosine Similarity

Given a set of training examples, $U = \{u^i, i = 1, \ldots, M\}$, we first apply feature extraction (as described in Sec. 4.3.1) to obtain their corresponding 1D signals $\vec{u}^i$ (Eq. 4.6). Using the same process, we transform an isolated test gesture into its corresponding query 1D signal, $\vec{q}$. We then classify $\vec{q}$ by locating the training vector $\hat{u}$ showing the Maximum Cosine Similarity (MCS):

$$\hat{u} = \arg \max_{\vec{u^i} \in U} \vec{q} \cdot \vec{u^i} \tag{5.1}$$

and assigning its label to $\vec{q}$.

In our case, vectors $\vec{u}$ and $\vec{q}$ are normalized vectors (Sec. 4.3), representing points on a unit hypersphere, and thus cosine similarity has the property of being in the range $[-1, 1]$; negative values place vectors farther away on this sphere, while positive – especially values near one – correspond to points that are very close.

5.1.2 Tree–based fast Nearest Neighbor

When vectors $\vec{u}$ and $\vec{q}$ are normalized vectors, the MCS problem is equivalent to a minimum Euclidean Distance problem, since:

$$\begin{aligned} \|\vec{q}-\vec{u}\|^2 = \|\vec{q}\|^2 + \|\vec{u}\|^2 - 2\vec{q}\cdot\vec{u} = 2\cdot(1-\vec{q}\cdot\vec{u}) \\ \Leftrightarrow \vec{q}\cdot\vec{u} = 1 - 0.5\|\vec{q}-\vec{u}\|^2 \end{aligned} \tag{5.2}$$

This formulation allows us to use fast search methods[1], such as the tree-based fast Nearest Neighbor (NN) method of Katsavounidis et al.[106], which offers an exact solution in nearly logarithmic time. This method operates much faster than the standard Full Search algorithm, which needs to scan all training examples and compute their distances to the query vector.

In more details, the first part (*initialization*) of this method involves recursive building of a binary tree. At each iteration, it splits the data, $u^{(i)} \in \mathbb{R}, i = 1,\ldots,m$, into two subsets, $C1$ and $C2$, using the K-Means algorithm [109]. It then computes the hyperplane $H(\tilde{w})$ that maximally separates the two cluster centroids, where $\tilde{w}$ is the coefficient vector of the hyperplane. By definition of $\tilde{w}$, if a vector q belongs to $C1$, then $d_s[\tilde{q}, H(\tilde{w})] < 0$, where $d_s[\tilde{q}, H(\tilde{w})] = \langle \tilde{q}, \tilde{w} \rangle$ denotes the *signed distance* of vector q from hyperplane $H(\tilde{w})$ and $\tilde{q}$ is the augmented vector $\tilde{q} = [1, q_1, \ldots, q_n]$. Thus, query points, q, can be classified as belonging to the *left* or *right* tree node child based on their signed distance, $d_s[\tilde{q}, H(\tilde{w})]$, and the algorithm continues recursively for $C1$ and $C2$.

During the second part (*searching*), a query vector, q, is continuously classified based on its signed distance, until a leaf node is reached (*depth–only stage* – DOS). Then the algorithm starts backtracking to the previous nodes and examines the rest of the vectors, since a global minimum is not guaranteed to lie at the first leaf node reached. Computational efficiency comes from pruning many vectors out of the search based on a lower bound, LB, of the Euclidean Distance, $d(q,u)$, since it holds that:

$$d(q,u) \geq LB = |d_s[\tilde{q}, H(\tilde{w})] - d_s[\tilde{u}, H(\tilde{w})]| \tag{5.3}$$

One can roughly measure the algorithm's efficiency by counting the number of *backtrackings*, B, i.e. the number of vectors with $LB < d_{min}$, where d_{min} denotes the running minimum distance at each node. Since all signed distances have been precomputed either during the depth–only stage or the training stage, computing LB requires only one subtraction.

[1] A brief overview of fast Nearest Neighbor algorithms is provided in Appendix A.

Terminating the search when the first leaf node is reached (Depth–only Search – *DOS*) has been found to work very efficiently and accurately on image compression tasks [106]. For such reasons, we also explore its performance as an approximate recognition method, additionally to exact fastNN.

5.1.3 Tree–based fast K Nearest Neighbor

It is also possible to extend the fastNN algorithm to perform K Nearest Neighbor search (*fastKNN–MCS*), which classifies a query vector through majority voting on the labels of the K nearest neighbors. This way, KNN is more robust to outliers and generally leads to higher recognition accuracy. Initialization phase is exactly the same as in standard fastNN. During the search phase, depth–only–search is performed until the first leaf is reached and then the algorithm starts backtracking. At each step, the K minimum distances are kept in a memory struct, while the maximum distance (out of the K) is used for pruning (Eq. 5.3). Clearly, fastNN becomes a special case of this algorithm, for $K = 1$.

In our experiments, we used $K = 5$, as we didn't notice significant improvements for larger values. In Chapter 8, we discuss computational complexity as a function of K.

5.2 Alternative recognition approches

A lot of approaches have been proposed for gesture recognition in the last 30 years. As described in Chapter 5, most of them are based in Dynamic Time Warping (DTW) [2], Maximum Cosine Similarity [6–8, 29, 42], Hidden Markov Models (HMMs) [30, 34, 35], Longest Common Subsequence (LCS) [10, 11], Conditional Random Fields (CRFs) [36–38], Dynamic Bayesian Networks (DBNs) [39] and Convolutional Neural Networks [40], presenting some variations and modifications.

In our experiments, we chose to compare to DTW with multi–dimensional observations (x, y) [2], as it is an exemplar approach, showing state–of–the–art results in a variety of time–series classification problems, including the special case of trajectory recognition which we use in this work. We also consider a model–based approach, using Support Vector Machines (SVM) [45]. In both cases, we put considerable effort in optimizing the computational efficiency of the classification process.

While DTW is in general expected to outperform MCS in terms of recognition accuracy, due to its nice warping properties, we are interested to know how much degradation one can expect when using MCS in the specific case of digits and letters, since it can serve

as a computationally lighter method. It is already known from the literature [42, 67] that MCS can achieve very high recognition results for stroke–based gestures.

One can also consider Hidden Markov Models (HMMs) [110] as an alternative approach but our experiments showed rather poor performance on most datasets with an average recognition rate of 64% for noiseless gestures, while DTW, SVM and our method performed in the upper 90 percentile range. Thus, we don't present any HMM results in the following, but we plan to investigate the usefulness of HMMs in the future, using standard implementations, such as the HTK toolkit [111].

5.2.1 Dynamic Time Warping

Dynamic Time Warping (DTW) [112] is a method to find the minimum–cost alignment between sample observations in two time series. Cost is measured as the sum of all local Euclidean distances between aligned samples [108]. Fig. 5.1 shows an example of DTW calculation and the optimal alignment, between a model time–series (e.g. a training example) of length M and a query time–series of length N.

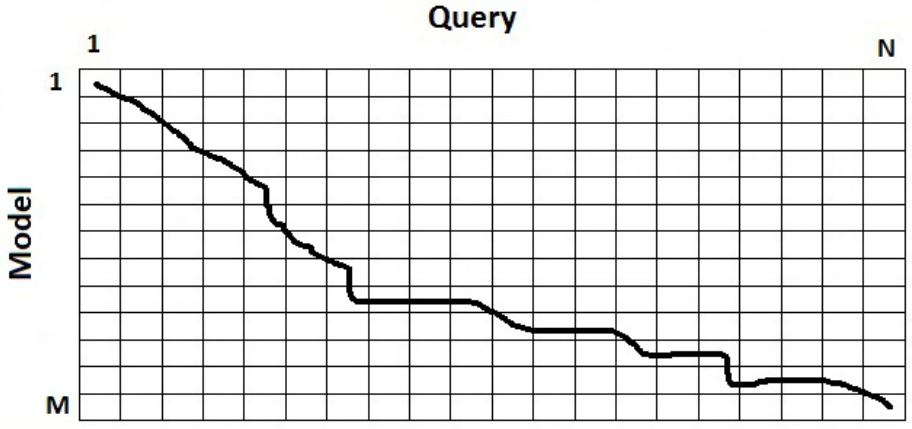


FIGURE 5.1: A typical example of DTW calcucation. The solid line shows the minimum–cost alignment.

Standard implementation of DTW is based on Dynamic Programming. Let $D(m, n)$ denote the DTW distance between the first m and n samples of model and query series respectively. Then, $D(m, n)$ can be computed recursively as:

$$D(m, n) = \min\{D(m-1, n-1), D(m-1, n), D(m, n-1)\} + d(m, n) \tag{5.4}$$

where $d(m, n)$ denotes the local distance between m–th and n–th sample, and $D(m, 0) = 0, D(0, n) = 0, \forall m, n$.

We also use a computationally efficient version of DTW (referred to as *DTW8* in this text), using our N–point representation (Sec. 4.3.1). In this case, we can also use the *Sakoe-Chiba band* restriction [113], which restricts the search area during the DTW calculation and the *LBKeogh* lower bound [107], which computes a lower bound on the global minimum distance and uses it to perform partial distance search.

While *LBKeogh* allows for exact search, it pre–assumes equal–length sequences and a restriction band, such as the *Sakoe-Chiba band*. Both techniques are approximations of the global minimum distance, although they have been shown to work well for a variety of problems. In Chapter 8, we discuss the computational efficiency provided by each of these techniques, supporting our arguments with thorough experimental results. Moreover, we present and evaluate a novel initialization method, which further improves the efficiency of DTW8, providing an improvement of 33% in execution time.

5.2.1.1 Probabilistic DTW

Alon et al.[2] proposed a probabilistic variant of DTW, which resembles a Hidden Markov Model (HMM), but uses a fixed transition matrix. Their model consists of Q states $q_i, i = 1, \ldots, Q$, while it is equally likely to remain at the same state or move to the next state. Each state, q_i, is associated with a multivariate Gaussian density function, (μ_i, Σ_i), linking the state to observations (e.g. hand coordinates), with μ_i and Σ_i) being the mean and the covariance matrix of the feature vectors observed in state i. These density functions are learned using a variant of the Baum–Welch algorithm [114], while the number of states, Q, is chosen manually.

During testing, a local cost is assigned to each pair of observation vectors and states, while the Viterbi algorithm is used to find the path of minimum total cost. Specifically, given a vector x and a state (q_i, μ_i, Σ_i), the local cost $d(x, q_i, \mu_i, \Sigma_i)$ is equal to the Mahalanobis distance:

$$d(x, q_i, \mu_i, \Sigma_i) = (x - \mu_i)^\top \Sigma^{-1} (x - \mu_i) \tag{5.5}$$

In our experiments, we refer to this DTW variant as "model–DTW". Since we didn't have any information on the number of states, Q, we manually chose $Q = 4$ which provided quite good results.

5.2.1.2 Normalized DTW

For our experiments, we explore five variations of DTW, as explained below:

- DTW: full DTW with all observations
- DTW–N: full DTW with normalized observations
- DTW8: resampled DTW with $N = 8$ and normalized observations
- model–DTW: Probabilistic DTW with all observations
- model–DTW–N: Probabilistic DTW with normalized observations

Standard DTW assumes that the trajectory points lie around some point of reference, which is subtracted (e.g. the face [2, 115]). In our implementation, we used the trajectory's centroid point, as the face location was not always available (e.g. in the Wii datasets). This way, the face is assumed to always lie at point $(0, 0)$, which still produces valid results.

In the cases of DTW–N, DTW8 and model–DTW–N, we further normalize the points, such that the corresponding vector has L_2-norm equal to 1, to ensure scale invariance, as described in Sec. 4.3.1. In the case of DTW8, such normalization makes sense in all applications. However, in the case of DTW and model–DTW, normalization may not be always a choice. Specifically, when used for gesture spotting in continuous streams, as in [2], model–DTW is built in a recursive fashion, using the results of the previous time step. Obviously, in such cases, normalization is only partially applicable, through a predefined global constant (e.g. face height [115]) instead of the locally adapted L_2-norm.

Our reason for exploring normalization is related to the effect of scaling on recognition accuracy. Specifically, we observed 3% lower recognition accuracy of DTW, compared to DTW8, in recognition of lower–case letters and severely distorted results in Kinect digit recognition. We further discuss this issue in sections 5.3.3 – 5.3.4, providing extensive experimental results.

5.2.2 Support Vector Machines

Support Vector Machines (SVMs) [45] refer to a Machine Learning algorithm which maximizes the separation margin between two classes, by solving a convex optimization problem. A query vector, q, is classified as positive or negative based on the sign of $f(q)$:

$$f(q) = \sum_{i=1}^{nSV} \alpha_i y_i K(s_i, q) + b \tag{5.6}$$

where b is a constant, s_i the so–called *support vectors* (chosen by the training examples), y_i their corresponding labels and α_i the weights of the support vectors. $K(s_i, q)$

denotes the kernel function, which controls feature mapping to high dimensional spaces, allowing for non–linear separations. The number of support vectors, nSV, depends on the difficulty of separation, data dimensionality and quality of feature extraction. More importantly, it is directly linked to computational efficiency, as Eq. 5.6 suggests [116].

In our work, we adopt the simple linear kernel, $K(s_i, q) = s_i \cdot q$, which allows for more efficient implementations, as:

$$f(q) = w \cdot q + b \tag{5.7}$$

where w denotes the maximum–margin separating hyperplane and can be precomputed as:

$$w = \sum_i \alpha_i y_i s_i \tag{5.8}$$

In this case, SVM requires only one computation of the inner product between w and q, while the number of support vectors does not affect computational efficiency anymore.

While SVMs were originally proposed for binary classification, it is also possible to perform multi–class classification by training C binary classifiers in a *1–vs–all* fashion. That is, each SVM hyperplane, $w_c, c = 1, \ldots, C$, separates class c from all other $C-1$ classes. The final classification result is derived by merging the results in some appropriate way [117]. Thus, 10 binary classifiers need to be trained for 10-digit recognition, while 26 classifiers are needed to recognize the 26 Latin letters.

In our work, we use a special form of *probabilistic SVM* [118], which offers a measure, p, for the quality of classification ($p \in [0, 1]$). Such probabilistic output is of great importance for rejection of invalid examples, as we will describe in Chapter 6.

For our experiments, we used the software package *LIBSVM* [119], a standard publicly available library, which offers efficient and robust implementations of various SVM–related tasks, including training and testing. We further optimized this library, achieving a speed–up of 5.8×, as described in Chapter 8.

5.3 Experimental results

5.3.1 Methods and Datasets

We ran thorough experiments, evaluating the performance of Maximum Cosine Similarity (MCS), which forms the basis of our gesture recognition system. We also experimented with Depth-Only-Search MCS (DOS-MCS), fastKNN–MCS Support Vector Machines (SVM), standard Dynamic Time Warping (DTW) and its low–complexity variant DTW8. We also explored using normalized observations in DTW, as this was

shown to improve recognition accuracy in certain cases. For our experiments, we use the datasets 1–6 (Table 3.1), i.e. the GreenDigits–EasyDigits (VGA video, digits), KinectR–KinectT (Microsoft Kinect camera, digits), Digits6D (Wii device, digits), Lower6D and Upper6D (Nintendo Wii device, lower and upper–case letters).

5.3.2 Effect of resampling parameter N

Our method allows for a downsampling parameter, N, denoting the number of trajectory points that will be used for gesture representation. This way, all feature vectors have equal length, $2N$, and time duration invariance is achieved. Moreover, downsampling deals with noise at high frequencies and results in higher computational efficiency. It is obvious that computational complexity scales linearly with N and thus we are interested to determine the *minimum* value that has good performance. While any value > 1 is allowed, it is preferred that $2N$ is a power of 2, allowing us to fully exploit the capabilities of modern CPUs for fast vector processing operations (e.g. Intel SSE™ and ARM Neon ™ SIMD instructions).

To this end, we varied the value of N and tested its effect on the GreenDigits dataset. We noticed that perfect recognition results could be obtained even for low values of N, e.g. $N = 4$. While this result is quite encouraging, it seems quite unprobable that only 4 points will suffice for recognition under real–world conditions. While in GreenDigits users wear a green glove, making hand detection easier and more accurate, in EasyDigits users sign with bare hands and short–sleeved clothes. Clearly, hand detection produces more noisy results in the second case, as it was already discussed in Sec. 4.1.3.

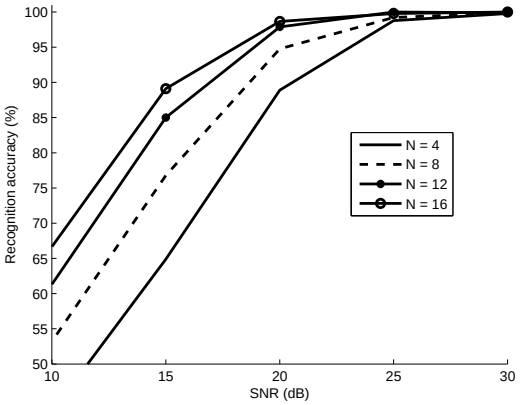


FIGURE 5.2: Recognition accuracy for noisy gestures (GreenDigits dataset) and various values of parameter N. For $SNR > 30$, recognition accuracy was close to 100%.

To resemble real–world conditions, we repeated our experiments on GreenDigits, under the presence of additive white Gaussian noise in the raw palm coordinate data (x, y). Fig. 5.2 presents recognition accuracy for different SNR values, showing that N becomes

more important for lower values of SNR (i.e. larger amount of noise). However, gesture trajectories become heavily distorted for $SNR < 20$ (Fig. 1.2) and they should probably be rejected by a gesture verification module. Thus, we finally choose $N = 8$, as the difference between the next value ($N = 12$) is quite small ($< 3\%$) for $SNR > 20$ dB. Moreover, we observed similar results on the digits6D dataset ($< 2\%$ drop between $N = 8$ and $N = 12$). Finally, choosing $N = 8$ leads to increased computational efficiency, as shown in Table 5.1.

TABLE 5.1: Execution time speed–up for MCS–fastNN (digits6D dataset)

N	**FullSearch–MCS**	**fastNN–MCS**
16	1×	21.2×
12	1.3×	26.2×
8	1.9×	34.0×

5.3.3 Performance on noisy data

Noise in gesture data may appear due to device flaws, user inexperience and temporary or permanent behavioural characteristics, such as anxiety, trembling hand or movement limitations. For this reason, we measured the accuracy of all methods under the presence of additive white Gaussian noise in the raw palm coordinate data (x, y). Tables 5.2 – 5.6 present our results for some typical SNR values (Eq. 4.4), for clarity of presentation.

We observe that both MCS and DTW achieve very high recognition accuracies in all datasets, especially for high SNR values. MCS seems to perform around 1.5% lower than the best result achieved, on average. Its fast approximation, MCS–DOS, is around $1-3\%$ worse, when compared to MCS. On the other hand, MCS–KNN ($K = 5$) presents almost the same results to MCS.

Although DTW seems to be the most accurate method, it is not the most robust. Specifically, we see that it achieves very low recognition accuracy for the KinectT dataset (43%). While this issue is not completely explored yet, we can assume it is due to scaling variations, since the accuracies for the normalized versions (DTW–N and DTW8) are the best achieved for this dataset. We observe a similar case in the lower6D dataset, where DTW achieves 95% (similar to MCS), while DTW8 achieves 97% and DTW–N 98.5%. In all other cases, we noticed similar performances between DTW and DTW–N.

Normalized model–based DTW performs around 2% worse than MCS on most datasets, while lack of normalization leads to severe drops for letters (7%) and mild drops for digits (1%), with the exception of KinectT, which once again shows severe degradation. Once again, data normalization proves to be very important for robustness. On Easy-Digits, model–DTW–N performs 10% worse than MCS, which seems to contradict the

TABLE 5.2: Recognition accuracy (%) for noisy gestures (EasyDigits)

Method/SNR	**40**	**30**	**20**	**14**	**10**
MCS	97.6	96.8	89.3	67.2	45.2
MCS–DOS	96.0	95.1	86.2	61.2	45.6
MCS–kNN5	97.8	97.4	89.4	67.6	46.9
DTW	97.9	98.3	97.8	91.4	80.7
DTW–N	97.8	98.0	97.5	85.7	65.0
DTW8	98.0	98.0	86.7	60.3	40.7
SVM	97.8	97.8	89.8	64.9	49.0
model–DTW	87.5	79.8	43.7	20.2	13.5
model–DTW–N	87.2	78.8	56.5	32.2	21.8

TABLE 5.3: Recognition accuracy (%) for noisy gestures (KinectT)

Method/SNR	**40**	**30**	**20**	**14**	**10**
MCS	94.9	95.0	93.8	91.0	86.3
MCS–DOS	82.2	81.9	81.0	76.9	73.5
MCS–kNN5	93.9	93.4	92.9	90.5	87.0
DTW	46.5	46.3	45.2	44.0	43.6
DTW–N	97.5	97.5	97.7	97.4	96.2
DTW8	96.7	97.5	97.1	95.4	86.0
SVM	95.0	94.7	92.5	88.9	85.9
model–DTW	32.0	32.0	32.0	31.2	28.1
model–DTW–N	94.0	93.0	91.0	86.5	73.9
MPLCS [10]	97	85	-	-	-

exceptional spotting performance reported in [2, 108]. However, those works used Dynamic Space-Time Warping (DSTW), an extension of Dynamic Time Warping (DTW), considering multiple hand candidates at each video frame and searching for the optimal path of hand locations over time. Moreover, [108] also reported that performance drops when only one hand candidate is used, thus explaining the drop we observed.

It is quite interesting that noise affects more camera data (EasyDigits), compared to Kinect and sensor based data. However, noise is inherent in most Computer Vision hand detection methods, while Kinect and Wii provide much cleaner input data. Tables 5.7 – 5.9 show the most severe cases of misclassification for noiseless data ($SNR = 40$ dB).

Probabilistic SVM performs similar to MCS for digits and around 1% worse for letters. Regarding the number of support vectors needed for separation of the classes, we observed that $40 - 72\%$ of the training examples are selected as support vectors, as shown

TABLE 5.4: Recognition accuracy (%) for noisy gestures (digits6D)

Method/SNR	40	30	20	14	10
MCS	98.8	98.7	98.8	98.5	97.3
MCS–DOS	97.2	97.1	96.7	95.9	94.6
MCS–kNN5	98.8	98.8	98.6	98.2	97.3
DTW	99.2	99.2	99.3	99.3	99.3
DTW–N	99.5	99.4	99.4	99.3	99.3
DTW8	99.3	99.3	99.3	99.0	96.8
SVM	98.6	98.7	98.8	98.3	97.5
model–DTW	97.7	97.7	97.6	95.3	90.3
model–DTW–N	98.3	98.3	98.3	96.4	92.4

TABLE 5.5: Recognition accuracy (%) for noisy gestures (lower6D)

Method/SNR	40	30	20	14	10
MCS	95.6	95.6	95.4	94.5	92.2
MCS–DOS	88.1	88.1	87.6	86.2	83.1
MCS–kNN5	94.8	94.7	94.2	93.4	90.9
DTW	95.2	95.2	95.1	94.8	94.2
DTW–N	98.5	98.8	98.5	98.1	97.5
DTW8	97.2	97.4	97.2	96.2	94.4
SVM	94.5	94.5	94.6	93.4	91.3
model–DTW	84.2	84.5	84.0	81.8	75.0
model–DTW–N	91.3	91.3	91.0	90.0	84.6

TABLE 5.6: Recognition accuracy (%) for noisy gestures (upper6D)

Method/SNR	40	30	20	14	10
MCS	97.5	97.5	97.4	96.3	93.8
MCS–DOS	94.8	94.8	94.5	92.8	89.9
MCS–kNN5	97.7	97.7	97.6	96.7	94.2
DTW	98.0	97.9	97.9	97.7	96.9
DTW–N	98.8	98.8	98.7	98.5	98.1
DTW8	98.1	98.2	97.8	96.9	94.7
SVM	96.7	96.6	96.6	95.8	93.1
model–DTW	89.0	89.0	88.7	86.7	80.9
model–DTW–N	96.0	95.9	95.5	93.5	88.1

in Table 5.10. We notice that EasyDigits and lower6D require a higher number of support vectors (close to 70%), while digits6D and upper6D much lower (close to 40%), implying easier separation and better structured data for the latter. While the number of support vectors is linked to computational efficiency in the general case [116], it is much less important in our work, due to the simple nature of the linear kernel used.

TABLE 5.7: Worst cases of misclassification (Kinect)

Digit	MCS	DTW	DTW8
0	8 (8.3%)	–	–
2	4 (16.7%)	–	4 (8.3%)
6	0 (8.3%)	0 (8.3%)	0 (8.3%)
7	1 (16.7%)	1 (16.7%)	1 (16.7%)

TABLE 5.8: Worst cases of misclassification (lower6D)

Letter	MCS	DTW8
k	h (16%)	–
h	n (10%)	–
t	e (28%)	–
y	–	x (6%)
a	–	d (8%)

TABLE 5.9: Worst cases of misclassification on the upper6D dataset

Gesture	MCS	DTW	DTW8
D	E,H, **P** (10.8%)	P (6.2%)	P (8.5%)
O	**C**, U (6.2%)	C (2.3%)	**C**, U (4.6%)
P	D (6.9%)	D (8.5%)	D (8.5%)
R	K (5.4%)	B (0.8%)	K (5.4%)
U	C,O,V (4.6%)	C,O,V (9.2%)	C,O,V (3.8%)

TABLE 5.10: Number of support vectors needed for classification)

Dataset	support vectors	# training examples	Percentage
EasyDigits	175	270	65%
Digits6D	200	500	40%
lower6D	754	1040	72.5%
upper6D	1378	3120	44%

5.3.4 Performance with fewer training examples

Our second group of experiments involves varying the number of training examples, in a leave–K–users–out cross–validation scheme. In each round, we considered K users for training set and $U - K$ users for validation, and averaged the results over 10 rounds (U denotes the total number of users). The two Kinect datasets were not included in this experiment since we didn't have access to explicit information about the users.

Tables 5.11 – 5.14 present our experimental results in terms of recognition accuracy for all datasets. We observe that in all cases, recognition accuracy increases with the number of users. Quite interestingly, digits are less sensitive to the amount of training examples, showing around $1 - 3\%$ difference between the two extreme cases ($K = 1$ and $K = U - 1$). On the other hand, recognition of lower and upper–case letters is more severely affected with few training examples ($10 - 12\%$ differences), perhaps due to larger variations caused by the larger number of classes (26 *vs* 10).

Exemplar DTW variations (DTW, DTW–N and DTW8) provide the best accuracies, but not high robustness. Once again, data normalization seems to be the key factor, as DTW8 performs similar to DTW for digits and much better ($5 - 10\%$) for letters, especially when one or two users are considered in the training set. On the other hand,

TABLE 5.11: Recognition accuracy (%) with varying number of training examples (EasyDigits)

Method/Users	1	3	5	7	9
MCS	95.6	97.1	97.5	97.3	97.3
MCS–DOS	93.6	95.9	96.0	96.4	96.0
MCS–kNN5	90.9	97.2	97.7	97.6	97.7
DTW	96.9	97.9	97.8	97.8	97.7
DTW–N	97.6	98.4	98.1	98.1	98.0
DTW8	96.9	97.6	97.9	98.2	98.0
SVM	65.6	97.2	97.6	97.9	97.9
model–DTW	71.4	85.8	88.8	88.4	88.3
model–DTW–N	72.7	82.5	85.4	85.9	87.0

TABLE 5.12: Recognition accuracy (%) with varying number of training examples (digits6D)

Method/Users	1	2	3	4	5
MCS	96.2	97.7	98.3	98.7	98.7
MCS–DOS	90.4	94.2	95.5	96.8	97.4
MCS–kNN5	94.2	97.3	98.4	98.4	98.7
DTW	97.5	99.1	99.3	99.3	99.2
DTW–N	97.9	99.2	99.4	99.5	99.5
DTW8	97.9	98.5	99.1	99.4	99.3
SVM	94.6	97.8	98.2	98.5	98.6
model–DTW	40.0	97.7	97.7	97.7	97.7
model–DTW–N	40.0	98.3	98.3	98.3	98.3

parameter estimation in model–based DTW variants is highly sensitive to the number of training examples, as depicted in our results for a low number of training users ($K < 3$). However, recognition accuracy improves significantly, even for 3 or 4 users. Performance of SVM is similarly affected by K, as expected by most model–based approaches.

5.3.5 Common recognition of digits and letters

An interesting scenario is to mix both digits and upper letters in the gesture alphabet, for increased user flexibility. Intuitively, we can expect some recognition errors, since some trajectories are very similar, such as digit–letter pairs "0 – O" and "1 – I". Indeed, we observed that both MCS and DTW highly confuse the above pairs, with accuracies close to 50%, as expected. Excluding "O" and "I" and keeping only "0" and "1", DTW achieved 97.6% and MCS 96.9%.

Table 5.15 shows the most important sources of errors for both MCS and DTW. We observe that the third most severe error is confusion between "2" and "Z". Most optical character recognition (OCR) algorithms face similar problems as well [120].

TABLE 5.13: Recognition accuracy (%) with varying number of training examples (lower6D)

Method/Users	1	2	3	4
MCS	82.7	91.2	93.6	95.5
MCS–DOS	74.9	84.1	86.5	88.0
MCS–kNN5	81.9	90.1	93.0	94.7
DTW	78.7	87.5	91.8	95.2
DTW–N	91.7	95.9	98.0	98.6
DTW8	89.5	94.3	96.1	97.2
SVM	80.9	90.5	93.0	94.4
model–DTW	56.9	72.6	80.9	84.2
model–DTW–N	74.0	87.0	90.2	91.4

TABLE 5.14: Recognition accuracy (%) with varying number of training examples (upper6D)

Method/Users	1	2	4	8	12
MCS	87.6	92.7	95.6	97.1	97.6
MCS–DOS	78.1	85.2	90.6	93.7	94.8
MCS–kNN5	83.7	91.2	94.8	97.3	97.9
DTW	82.4	90.5	95.1	97.2	98.1
DTW–N	95.4	97.2	98.2	98.6	98.7
DTW8	91.1	94.8	96.7	97.7	98.1
SVM	82.6	90.4	94.5	96.2	96.7
model–DTW	55.7	72.5	82.1	87.7	89.1
model–DTW–N	71.3	84.7	91.8	94.8	96.0

5.4 Evaluation based on information gain

5.4.1 Definition of information gain

Although average recognition accuracy is the standard evaluation method in most classification works, we believe it provides a very coarse view of a system's actual performance, since it takes into account only the correct predictions. However, the structure of the confusion matrix may provide valuable information about the severity of misclassification errors. Moreover, average recognition accuracy ignores the prior probabilities of categories, assuming a uniform prior, although this assumption is rarely valid. For example, distribution of letters may vary based on the language (e.g. English vs French) or the nature of a text (e.g. sports vs news). Ignoring the underlying distribution may lead to biased results: in the extreme case of a class that has very low occurrence rate, a classifier may (falsely) present very good accuracy by avoiding this low–probability event. For such reasons, we also explore and evaluate a novel performance measure based on the information gain between the actual gesture signed by the user and the predicted gesture interpreted by the recognition system.

TABLE 5.15: Worst cases of misclassification for MCS, DTW and DTW8 on the Alphanumeric dataset

Gesture	MCS	DTW	DTW8
C	L, U (5.4%)	L, U (6.2%)	L, U (6.1%)
D	P (7.7%)	P (6.2%)	7, B, P (15.4%)
K	R (4.6%)	E (4.6%)	A, B, E, R (12.2%)
P	D, M (8.5%)	D (8.5%)	D (11.5%)
R	K (4.6%)	B (0.8%)	B, K (7%)
U	C, V (3.8%)	C (7.7%)	C, V (6.2%)
Y	T (0.8%)	T (4.6%)	E, H, T, X (8.6%)
Z	2 (4.6%)	2, F (3.8%)	2, F (13.8%)
0	6 (3.3%)	6, C (3.3%)	6 (3.3%)
2	Z (23.3%)	Z (13.3%)	Z (26.7%)
4	9, Q (5%)	9 (5%)	9 (3.3%)
6	0, 4, Q, X (8.3%)	0, Q, X (5%)	0, Q, X (5.1%)
7	D (5%)	D (5%)	D (3.3%)

One way to view a gesture recognition system is as a standard communication system, where a *transmitter* (user) wants to send a *message* X (gesture) derived from a *discrete source alphabet* $\mathcal{X}$ (gesture classes) to a *receiver* (computational system) through a *noisy communication channel* (capturing device). The channel is characterized by the transition matrix $p(y|x) = Prob\{Y = y|X = x\}$, where random variable Y denotes the received message. Assuming a discrete probability distribution over the alphabet $\mathcal{X}$, with probability mass function (*pmf*) $p(x)$, the entropy of random variable X is defined as:

$$H_b(X) = -\sum_{x\in\mathcal{X}} p(x)\log_b p(x) \tag{5.9}$$

and expresses the amount of a priori uncertainty on the value of X.

The maximum possible value of $H_b(X)$ is $\log_b |\mathcal{X}|$, achieved when X follows a uniform distribution ($|\mathcal{X}|$ denotes the cardinality of a set $\mathcal{X}$). When the base of logarithm, b, is equal to 2, entropy is expressed in bits, reaching a maximum value of $H_2(X) = 1$ bit for a binary alphabet ($|X| = 2$). Unless otherwise stated, we will assume a logarithm base of 2, measuring $H(X) \equiv H_2(X)$ and all other information theoretic quantities in bits.

The conditional entropy of X given Y is defined as:

$$H_b(X|Y) = -\sum_x p(y)\cdot H_b(X|Y = y) \tag{5.10}$$

where

$$H(X|Y = y) \equiv H_2(X|Y = y) = -\sum_x p(x|y)\cdot \log_2 p(x|y) \tag{5.11}$$

Finally, the mutual information (in bits) between random variables X and Y is defined as:

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{5.12}$$

that measures the decrease in uncertainty of X caused by the knowledge of Y. Additionally, we define the *normalized mutual information*, $\hat{I}(X;Y)$ as:

$$\hat{I}(X;Y) = I(X;Y)/H(X) = 1 - \frac{H(X|Y)}{H(X)} \tag{5.13}$$

In gesture recognition, $\hat{I}(X;Y)$ expresses the percentage of a priori uncertainty (on the transmitted gesture X) explained by the knowledge of the recognized gesture, Y. An optimal system should have $\hat{I}(X;Y) = 1$ (or 100%).

In fact, $\hat{I}(X;Y)$ is equivalent to $I_b(X;Y)$ for $b = 2^{H_2(X)}$, where b is the cardinality of the uniformly distributed random variable with the same uncertainty. Therefore, it relates to the per symbol cardinality of the Asymptotic Equipartition Property *(AEP)* set. In that case, $H_b(X) = 1$ and obviously $I_b(X;Y) \leq 1$.

5.4.2 Experimental results

Evaluation based on the normalized mutual information, $\hat{I}(X;Y)$, requires knowledge of system's transition matrix, $p(y|x)$, and the prior distribution of input gestures, $p(x)$. Learning $p(y|x)$ is achieved by computing a confusion matrix through cross–validation, as conducted for our experiments in Sec. 5.3.3. Given the different results of Tables 5.2 – 5.6, transition matrix varies based on the type of sensor and the SNR of additive Gaussian noise (i.e. the channel parameters).

Tables 5.16 – 5.17 show mutual information, $I(X;Y)$, for MCS and DTW8 classifiers, respectively, as well as for various SNR values [2]. In all cases, a uniform prior distribution $p(x)$ is assumed for digits ($H(X) = \log_2 10 = 3.3219$ bits) and letters ($H(X) = \log_2 26 = 4.7004$ bits). As expected, information is higher for higher SNR, i.e. when gesture is written more accurately.

Tables 5.18 – 5.19 show corresponding normalized mutual information values, $\hat{I}(X;Y)$, for MCS and DTW8 respectively. Comparing to Tables 5.2 – 5.6, we see that $\hat{I}(X;Y)$ is always lower than average recognition accuracy, providing a more reliable estimate of system's future performance, since it takes into account both the correct predictions as well as the severity of misclassification errors.

[2] Mutual information values for $SNR = 40$ were almost the same as for $SNR = 30$ and are thus not shown for clarity of presentation.

TABLE 5.16: Information gain for noisy gestures (MCS)

Dataset/SNR	30	20	14	10
EasyDigits	3.15	2.72	1.75	0.97
KinectT	3.10	3.07	2.98	2.69
digits6D	3.24	3.25	3.22	3.15
lower6D	4.47	4.45	4.40	4.25
upper6D	4.55	4.52	4.49	4.39

TABLE 5.17: Information gain for noisy gestures (DTW8)

Dataset/SNR	30	20	14	10
EasyDigits	3.21	2.55	1.55	0.84
KinectT	3.16	3.17	3.03	2.83
digits6D	3.27	3.27	3.23	3.15
lower6D	4.56	4.54	4.51	4.37
upper6D	4.58	4.56	4.51	4.38

TABLE 5.18: Normalized information gain (%) for noisy gestures (MCS)

Dataset/SNR	30	20	14	10
EasyDigits	94.8	81.9	52.6	29.2
KinectT	93.4	92.4	89.8	81.0
digits6D	97.7	97.7	96.8	95.0
lower6D	95.0	94.7	93.5	90.5
upper6D	96.8	96.2	95.4	93.3

TABLE 5.19: Normalized information gain (%) for noisy gestures (DTW8)

Dataset/SNR	30	20	14	10
EasyDigits	96.6	76.9	46.7	25.3
KinectT	95.1	95.5	91.2	85.1
digits6D	98.4	98.4	97.2	94.7
lower6D	97.0	96.5	96.0	93.0
upper6D	97.5	97.0	96.0	93.2

Regarding prior distribution of alphabet $\mathcal{X}$, $p(x)$, it seems natural to assume almost uniform distribution for digits, unless other prior knowledge is available[3].

On the other hand, distribution of letters depends on the language. To this end, we repeated our experiments using letter frequencies for three widely spoken languages (English, Spanish and French) and compared to Tables 5.18 – 5.19. Our experiments revealed that $\hat{I}(X;Y)$ degrades around 2.5% for lower letters and 1% for upper letters, on average, for all three languages.

5.5 Discussion

In this chapter, we explored isolated recognition for trajectories of digits and letters. Our approach is based on Maximum Cosine Similarity (MCS) and a tree–based fast Nearest Neighbor algorithm [106]. In our experiments, conducted on three publicly available databases, we explored various parameters, such as recognition with a varying number of training users/examples, performance on noisy data and common recognition of digits and letters. Moreover, we explored information gain as a new measure to evaluate recognition accuracy, that we believe depicts a better picture about performance of a gesture recognition system.

[3] According to Benford's law for long lists of numbers, the probability that "the first digit of a number is a" is equal to $\log_{10}\frac{a}{a+1}$. However, the same law predicts that the probability of observing digit a in position q is approximately 0.1 for $q \neq 1$.

Hopefully, the results of this chapter will provide more insight to the results presented next, regarding rejection of invalid examples (Chapter 6) and continuous gesture spotting (Chapter 7), leading to a more complete understanding of the gesture recognition problem.

Chapter 6

Gesture verification

Contents

6.1 Motivation

Gesture verification is the process of establishing whether a gesture is indeed an instance from a predefined vocabulary or an invalid gesture (*false positive*). In this work, we consider three main types of invalid gestures, namely 1) *out–of–vocabulary*, 2) *noisy* gestures and 3) *random* movements (Sec. 1.4). In short, out–of–vocabulary gestures refer to shapes and symbols that have a meaning in the proper context, but are not

included in system's vocabulary (e.g. letters instead of digits). Noisy gestures refer to valid gestures with some additive Gaussian noise, where the goal is to reject highly noisy gestures (Fig. 1.2). Finally, completely random hand coordinates are examples of signals that may be fed to a gesture recognition system in a variety of uncontrolled scenarios during system's inactivity periods (e.g. walking, drinking water, etc.).

While rejection of invalid examples is obviously important to establish any system's reliable (or robust) performance, it is also important for computational efficiency. Specifically, random or out–of–vocabulary inputs typically compromise the effectiveness of fast Nearest Neighbor search (*fastNN*), causing too many backtrackings [121]. To deal with this problem, we propose setting a threshold on the maximum number of backtrackings allowed for classification of a query gesture. By doing so, query gestures that cause too many backtrackings will be rejected as invalid.

Fig. 6.1 shows the average number of backtrackings, $\bar{B}$, on a system trained on digits. We observe that noisless digits present $\bar{B} \approx 17$. Similarly, digits with low noise ($15-40$ dB) require comparable backtrackings ($\bar{B} \approx 20$), although $\bar{B}$ increases with noise. Finally, upper–case and lower–case letters (*i.e. out–of–vocabulary gestures*) cause a significantly higher number of backtrackings ($\bar{B} \approx 100$).

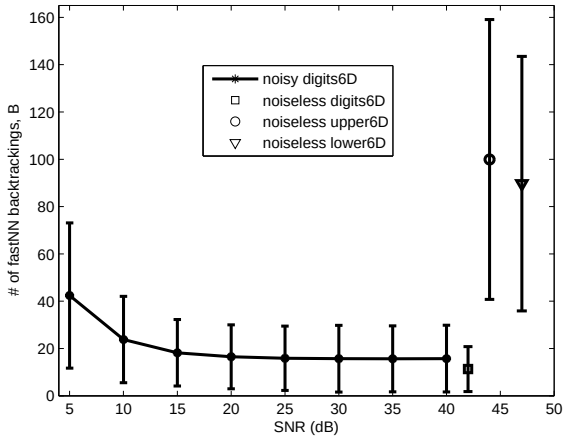


FIGURE 6.1: Average number of backtrackings, $\bar{B}$, on a system trained on digits, for various categories of input gestures. Error bars show standard deviation. Digits with higher noise (i.e. lower SNR) require more – but comparable – backtrackings, $\bar{B}$. Out–of–vocabulary inputs (upper/lower letters) require significantly higher $\bar{B}$, even without noise.

In the rest of this chapter, we present our approach in detail and explore its performance, compared to other standard methods for gesture verification. Moreover, we explore in depth the relationship between the performance of fastNN algorithm and the resulting number of backtrackings. In all cases, we conduct thorough experiments on synthetic and real datasets.

In short, the major contributions of this chapter are:

- Proposing a novel method for gesture verification, based on cosine similarity and the number of fastNN backtrackings required to classify an unknown gesture input.
- Introducing a new information–theoretic metric to evaluate gesture verification methods.
- Exploring the relationship between the number of fastNN backtrackings and the probability of inlier. This property leads to high computational efficiency, as one may constrain the search time up to a certain number of backtrackings, rejecting inputs that exceed a predetermined threshold as invalid.
- Exploring and proving a weak relationship between computational efficiency and recognition accuracy, through the Minimum Backtrackings classifier.
- Exploring various aspects of gesture verification, such as effect of noise and discrimination between digits and letters.

Algorithm 1 Isolated gesture recognition with rejection of invalid examples

Input: hand coordinates x, y
resampling parameter N
thresholds T_B, T_P
Output: Label C of the gesture

1: Resample x, y and form 1D vector v as described in Sec. 4.3.1
2: $(B, P, c) \leftarrow$ fastNNsearch(x, y, N) ▷ *B: # backtrackings, P: similarity score, c: predicted class label*
3: $C \leftarrow$ CheckRule3(B, P, T_B, T_P, c) ▷ Alternatively one can use Rule1 or Rule2
4: **return** C ▷ C is the final recognition result

Rule definitions
CheckRule1: Gesture is valid if $P > T_P$
CheckRule2: Gesture is valid if $B < T_B$
CheckRule3: Gesture is valid if $B < T_B$ AND $P > T_P$

6.2 Our approach

6.2.1 Rejection of invalid gestures

Typically, we can expect an out–of–vocabulary gesture to present a very low similarity score, P, and thus it can be rejected based on an appropriate threshold, T_P [29] (*Rule1* in Algorithm 1). However, since random or out–of–vocabulary inputs typically compromise

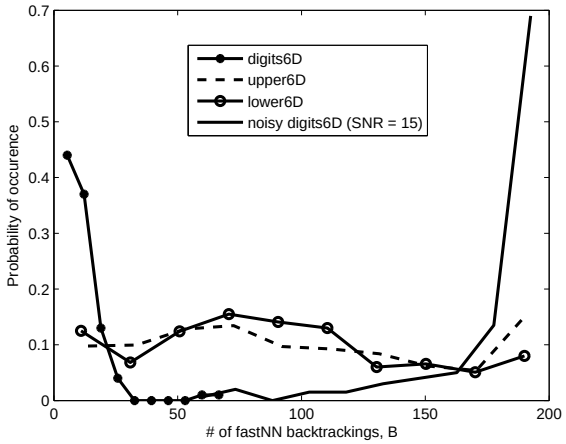


FIGURE 6.2: Distribution of backtrackings on a system trained on digits, for various categories of input gestures. While some overlap typically exists, it is still possible to keep most of the noisless digits, together with few lower and upper–case letters.

the effectiveness of fastNN, causing too many backtrackings [121], we propose setting a threshold, T_B, on the maximum number of backtrackings, B, allowed for classification of a query gesture. If $B > T_B$, the gesture should be rejected as invalid (*Rule2* in Algorithm 1). As Fig. 6.2 shows, it is possible to choose T_B through cross validation, achieving a satisfactory trade–off between percentage of accepted valid gestures (True Positive Rate - TPR) and accepted invalid gestures (False Positive Rate - FPR). Finally, we can expect that using both similarity score and number of backtrackings can result in better results (*Rule3* in Algorithm 1). In practice, using the number of backtrackings as an "early rejection" criterion serves dual purpose: we improve system's reliability, while limiting computational complexity.

TABLE 6.1: Confusion matrix

	Prediction:Positive	**Prediction:Negative**
Reality:Positive	True Positives	Type I errors (False Negatives)
Reality:Negative	Type II errors (False Positives)	True Negatives

6.2.2 Measuring performance

Given a set of query positive and negative gestures and a fixed threshold, T, any rule will predict a label for each gesture. That label can be either *positive* (i.e. the gesture is *valid*) or *negative* (i.e. the gesture is *invalid*). A comparison to the ground truth labels can result in the confusion matrix (Table. 6.1), which shows the distribution of correct and wrong decisions. In such analysis, four cases can appear:

- True Positive (TP): a truly valid gesture is classified as valid
- True Negative (TN): a truly invalid gesture is classified as invalid

- False Negative (FN): a truly valid gesture is classified as invalid (Type I error)
- False Positive (FP): a truly invalid gesture is classified as valid (Type II error)

6.2.2.1 Precision and Recall

Some typical measures for performance evaluation of decision systems include *Precision* and *Recall*. Precision depicts the probability that a classified positive gesture is truly positive:

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

while Recall depicts the probability that a truly positive gesture will be classified as positive:

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

Typically, there is a trade-off between Precision and Recall, depicted by the *Receiver Operating Characteristic (ROC curve)*, produced through varying T for a wide range of values. A combining measure is the *F1 score*, computed as:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6.3}$$

$F1$ score provides a more compact overview, being equal to 1 for a perfect system and equal to 0 in the worst case. A standard way of comparing two ROC curves is the point of *Equal Error Rate – EER*, where $Precision = Recall$.

6.2.2.2 Evaluation based on the information gain

We also evaluate the performance of a verification rule R based on the information gain provided by R over the naive rule of always selecting the class of maximum a priori probability[1]. Specifically, we measure the information gain, $I(X;Y)$ (Eq. 5.12) and the normalized mutual information, $\hat{I}(X;Y)$, (Eq. 5.13), as in Sec. 5.4.

[1]This evaluation method was inspired by the *clever rain man* problem, given as homework in the Information Theory class (EE565a) taught by Prof. Zheng Zhang at the University of Southern California, 1992.

6.2.3 Alternative methods

6.2.3.1 Dynamic Time Warping

As an alternative, we consider Dynamic Time Warping (DTW), which provided optimal recognition results for trajectory recognition, but on a high computational cost (Sec. 5.3). Based on Sec. 5.2.1.2, we explore two variations of data preprocessing before using DTW, as explained below:

- DTW: standard DTW with all observations (not resampled, not normalized)
- DTW8: resampled DTW with $N = 8$ and normalized observations

In both cases, the mean was subtracted from hand coordinates, centering them around $(0, 0)$. In the general case, coordinates are centered around a fixed point of reference, such as the user's face [108].

Please note that DTW (standard DTW with all observations) is the version we used in chapter 5, where it showed excellent behaviour in most datasets, but faced some problems when used on the Kinect dataset (mainly due to scaling). However, we choose to use it here since it is the main algorithm used in previous gesture spotting works [2, 108].

6.2.3.2 Probabilistic Support Vector Machines

Given a query vector q, probabilistic Support Vector Machines (SVMs) [45] (Sec. 5.2.2) estimate the probability $p(c) = Prob[Class = c|q], c = 1, \ldots, C$ for all C classes. We then use $p(c)$ to filter out invalid examples of very low probability. Although one–class SVMs [122] might be more suitable for verification tasks, we opted to test probabilistic SVMs due to their good performance in isolated recognition (Sec. 5.3.3 – 5.3.4). Additionally, one–class SVMs require training a different model exclusively for verification.

6.2.3.3 Mahalanobis distance and probability of inlier

A standard generic approach of testing for outliers is based on statistical hypothesis testing. Such tests check the likelihood of the *Null Hypothesis*, H_0, i.e. the probability that an observed sample is an inlier, based on some measured quantity (known as *test statistic*). In this work, our test statistic is based on the Mahalanabis distance, which is a generalization of the standard Euclidean distance.

Given a set of positive examples, $X = \{x_i\}, x_i \in \mathbb{R}^N, i = 1, \ldots, M$, with estimated mean μ and covariance matrix S^{-1}, the squared Mahalanabis distance, $D^2(X, q)$, of a query vector $q \in \mathbb{R}^N$ to set X is defined as:

$$D^2(X, q) = (q - \mu)^\top S^{-1}(q - \mu) \tag{6.4}$$

Due to noise in data acquisition, the observed Mahalanobis distance d^* of q to X may drift from its true value. The Null Hypothesis, H_0, assumes that q is an inlier ($D^2(X, q) = 0$), while the alternative hypothesis, H_A, assumes that q is an outlier ($D^2(X, q) > 0$). Standard hypothesis testing checks the *P–value*, which measures the probability of observing data towards the direction of the alternative hypothesis, assuming that the Null hypothesis is true. Typically, one accepts H_0 if the P–value is larger than 5%, otherwise accepts the H_A.

In our work, P–value measures the probability that $D^2(X, q)$ is larger than the observed Mahalanobis distance d^*, assuming that q is an inlier. Thus, we define the probability of inlier, P_0, as:

$$P_0 = Prob\{D^2(X, q) > d^*\} = 1 - F(d^*) \tag{6.5}$$

where $F(x)$ denotes the cumulative distribution function (*cdf*) of Mahalanobis distances $D^2(X, q)$.

When set X and a set of query vectors $Q = \{q_i\}, i = 1, \ldots, L$ are multivariate normally distributed in N dimensions, $\lambda D^2(X, q)$ follows the Fisher—Snedecor distribution (or F–distribution) with N and $M - N$ degrees of freedom, where

$$\lambda = \frac{M(M - N)}{N(M + 1)(M - 1)} \tag{6.6}$$

The cumulative distribution function (*cdf*) of the F–distribution is:

$$F(x; \alpha, \beta) = B(\frac{\alpha x}{\alpha x + \beta}; \frac{\alpha}{2}, \frac{\beta}{2}) \tag{6.7}$$

where B indicates the regularized incomplete beta function:

$$B(x; \alpha, \beta) = \int_0^x t^{a-1}(1 - t)^{b-1} dt \tag{6.8}$$

Cdf values of the F-distribution can be computed numerically using statistical packages, such as Matlab$^{\text{TM}}$.

If P_0 is higher than a threshold, T_H, then the Null hypothesis is valid, and q is accepted, as an inlier; otherwise, q is rejected, as an outlier. Through varying T_H, a ROC curve can be generated (Sec. 6.2.2).

In our approach, we first perform a fastNN search to assign a query vector to a gesture category and then compute its Mahalanobis distance from the set of training examples that belong to that same category. This way, we can compute a more robust estimate of P_0, since the training set contains instances from a single distribution instead of mixtures of different distributions.

6.3 Experiments on gesture datasets

As a first step, we performed experiments using real gesture datasets (Sec. 5.3.1). Specifically, we used the datasets 1–6 (Table 3.1), i.e. the EasyDigits (VGA video, digits), KinectR–KinectT (Kinect camera, digits), KinectNonDigits (mathematical symbols), Digits6D (Wii device, digits), Lower6D and Upper6D (Wii device, lower and upper–case letters).

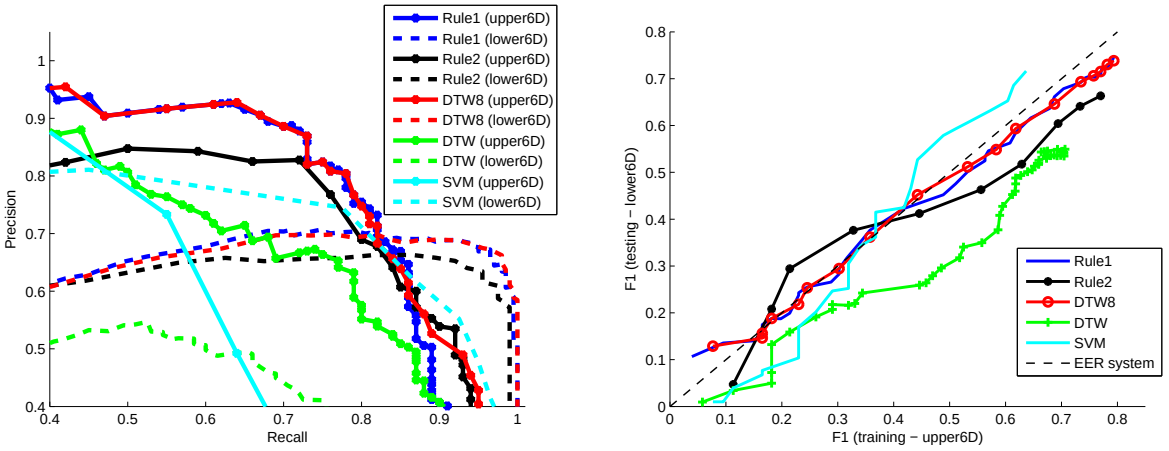
6.3.1 Case 1: out–of–vocabulary gestures

6.3.1.1 Separating digits from letters

We split the digits6D dataset into three subsets:

- *digitsTree*, used for building the fastNN tree (2 users – 200 examples)
- *digitsR*, used for invalid gesture classifier training, i.e. collecting statistics and determining appropriate thresholds regarding the number of backtrackings and cosine similarity scores (1 user – 100 examples)
- *digitsT*, used for testing/evaluation

Based on misclassification errors between digits and letters (Sec. 5.3.5), we excluded the letters “I” , “O”, “Z”, “i”, “o” and “z” from our analysis. During training, we ran MCS for digitsR (100 positive examples) and upper6D (3120 negative examples) and kept the resulting similarity scores and number of backtrackings as features. During evaluation, we performed the same process for digitsT (200 positive examples) and lower6D (1200 negative examples). Please note that, while positive examples were taken from the same distribution during statistics collection and evaluation, we opted to use two fundamentally different negative sets; upper letters for statistics collection and lower letters for evaluation, since our purpose is to determine an efficient method to reject *all* out–of–vocabulary gestures, and not only those used when determining classifier thresholds. Furthermore, splitting datasets in a user–independent mode decreases any bias, but makes the verification part more challenging.



(a) (b)

FIGURE 6.3: (a) Precision–Recall trade–off for rejection of upper (training) and lower (testing) letters by a system trained on 6D digits. Rejection rule based on the number of backtrackings (*Rule2*) performs around 3% worse than classic similarity-based *Rule1* and normalized–resampled DTW8. Standard un–normalized DTW performs worse than DTW8. (b) Generalization performance based on the F1 scores for fixed thresholds. Rule2 and DTW8 perform close to the ideal system of slope 1.

ROC curves Finally, we computed the Receiver Operating Characteristic (ROC) curve, which shows the trade–off between Precision and Recall for each method. Although the two classes (valid and invalid) are heavily skewed, with numbers of examples differing by an order of magnitude, Precision–Recall analysis deals effectively with such issues. As we note in Fig. 6.3–a, MCS *Rule1* and DTW8 present similar behaviour on the training set, with EER close to 0.79. On the other hand, *Rule2* performs around 3% worse ($EER \approx 0.76$), while un–normalized DTW shows the worst behaviour ($EER \approx 0.69$).

Due to the different nature of training and testing sets (upper versus lower–case letters), perfect generalization cannot be expected. Indeed, DTW8 and Rule1 perform approximately 9% worse ($EER \approx 0.70$), *Rule2* 12% worse ($EER \approx 0.64$) and DTW 15% worse ($EER \approx 0.52$). Fig. 6.3–b shows the F1 scores achieved for fixed thresholds between training and testing sets. We observe that *Rule1* and DTW8 present the best behaviour, performing close to the ideal line of slope $\lambda = 1$, while Rule2 follows closely. Based on the above results, MCS *Rule1* and DTW8 seem to guarantee better generalization on unknown vocabularies, compared to Rule2 and un–normalized DTW.

Overall, the above results suggest that it is possible to achieve high recall (≥ 0.9) with satisfactory precision (> 0.6), proving that many invalid gestures can be eliminated during gesture verification. While such elimination affects only system's reliability when DTW is used, it has the added benefit of improving computational efficiency in MCS systems, as one can set an upper bound on the number of fastNN backtrackings. Fig. 6.4 presents Precision and Recall as functions of the threshold T_B. As we see, the EER

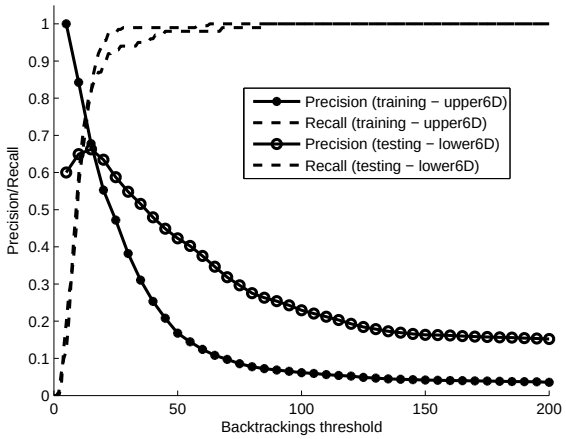


FIGURE 6.4: Effect of thresholding the number of backtrackings at various values. While Recall remains almost unchanged, Precision drops at the testing set, due to differences between upper and lower–case letters. The EER points can be achieved for a low threshold ($T_B \approx 20$), resulting in increased computational efficiency.

point is achieved for $T_B \approx 20$, i.e. up to 10% of the training examples in the fastNN tree can be checked before the gesture is classified as valid or invalid.

Information gain To confirm the above results, we also explored our alternative evaluation method, based on the information gain (in bits), $I(X;Y)$, provided by a rule over the naive rule of always selecting the class of maximum a priori probability (Sec. 6.2.2.2). Fig. 6.5 shows $I(X;Y)$ for all six rules and various values of the thresholds, with peaks indicating the optimal threshold choices. Information gain is higher for the testing set (lower6D) mainly because of higher initial entropy too ($H(Y) \approx 0.6$ bits versus 0.2 for the training set), which offers greater chances for improvement. The low values of $H(Y)$ confirm that the two classes are highly skewed, as two balanced classes would show $H(Y) = 1$. Once again we observe that it is possible to stop the fastNN search at a low number of backtrackings (20 out of 200), achieving both computational efficiency and high gesture verification results.

Rule3 Additionally, we evaluated *Rule3*, which takes into account both the number of backtrackings and the cosine similarity score. In this case, we measured both information gain and F1 scores for various parameter combinations, obtaining the 3D plots of Fig. 6.6. As we observe, both measures present similar behaviours, reaching their maxima for high cosine similarity values and low number of backtrackings. It is worth noting that *Rule3* performs better than Rule1, although the difference is very small. However, Rule3 allows for better generalization, as it is quite flat in its peaks. In general, information gain seems to be more promising as an evaluation technique, showing in a clear way that

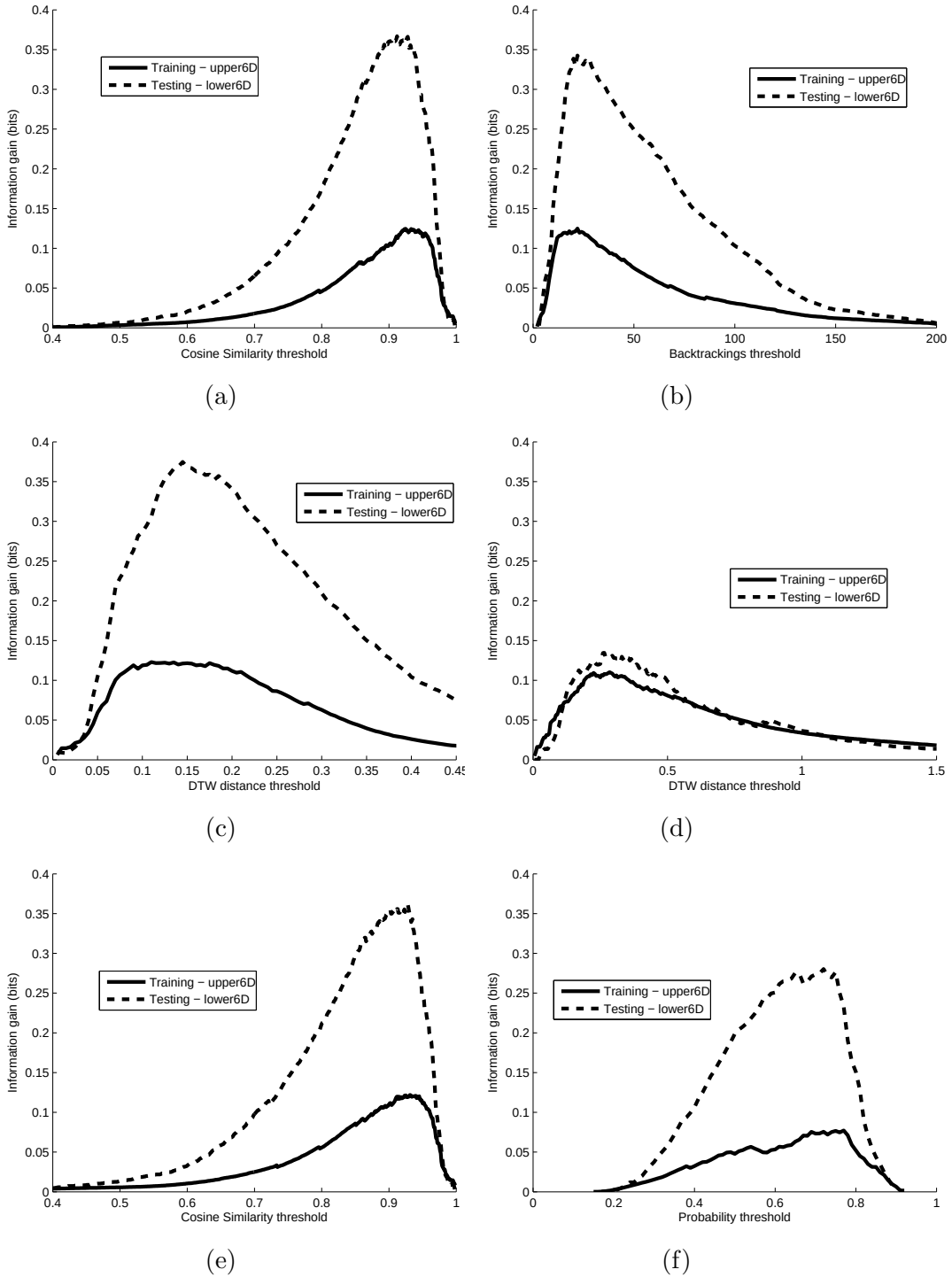


FIGURE 6.5: Evaluation of gesture verification rules by a system trained on 6D digits, based on the information gain they provide over the prior class probabilities. The goal is to reject upper (training) and lower (testing) letters. (a) MCS *Rule1* (b) MCS *Rule2* (c) DTW8 *Rule1* (d) DTW *Rule1* (e) MCS–DOS *Rule1* (f) SVM *Rule1*

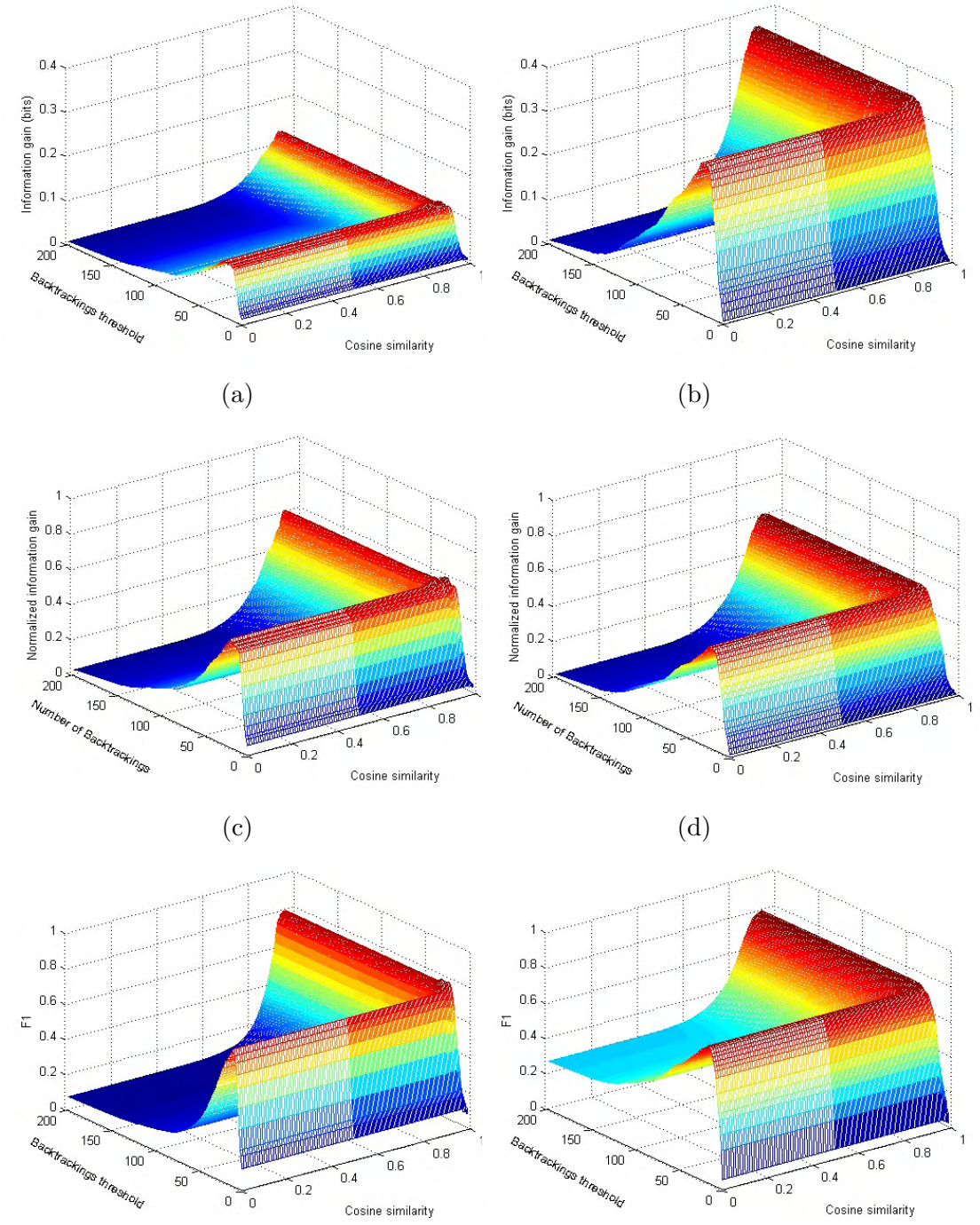
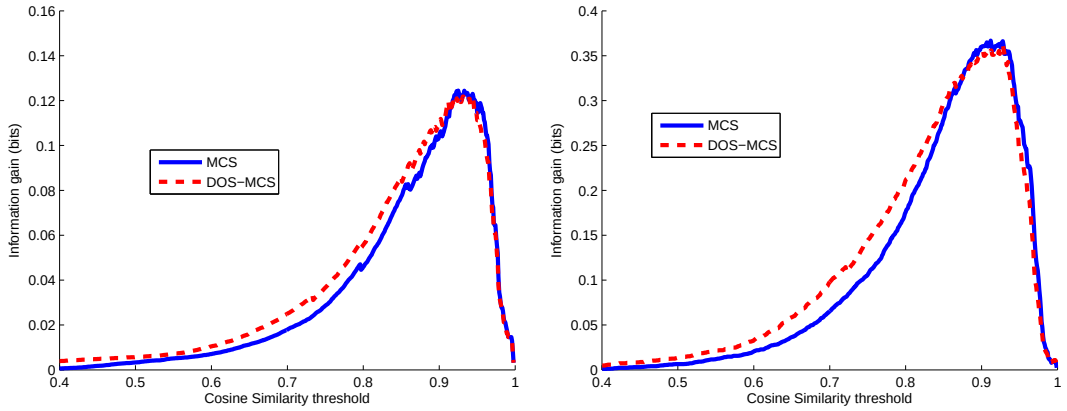


FIGURE 6.6: (a-b): Evaluation of gesture verification Rule3 by a system trained on 6D digits, based on the information gain they provide over the prior class probabilities. *Backtrackings threshold* indicates the number of backtrackings allowed during the fastNN search. The goal is to reject upper (a) and lower (b) letters in a system trained on digits. (c-d) Corresponding normalized information gains for the same evaluation. (e-f) F1 score for the same evaluation.

the two datasets (lower6D and upper6D) are different, as confirmed by Fig. 6.3. On the other hand, normalized information gain and F1 score reach similar maximum values in both cases.

Depth–only Rule1 Finally, we evaluated the effect of using the Depth–Only–Search version of fastNN along with MCS *Rule1* (MCS–DOS). As described in Chapter 5, Depth–Only–Search provides increased computational efficiency at the cost of slightly reduced isolated recognition accuracy (Sec. 5.3.3 – 5.3.4).

Fig. 6.7 shows information gains of MCS–DOS compared to standard MCS. We observe that MCS–DOS performs slightly worse than MCS. This observation agrees for the most part with the results of sections 5.3.3 – 5.3.4, where we saw that MCS–DOS performs worse for digits ($2-12\%$ drop in terms of recognition accuracy) and letters ($3-7\%$ drop). These two results indicate that gesture verification is possible – up to some level – during the depth–only stage of fastNN search, while allowing more fastNN backtrackings increases isolated recognition accuracy.



(a) (b)

FIGURE 6.7: Evaluation of gesture verification *Rule1* variations (standard MCS and MCS–DOS) in a system trained on 6D digits, based on the information gain they provide for (a) upper–case and (b) lower–case letters.

Indeed, as we observe in Fig. 6.8, performance of Rule1 is almost independent of the number of backtrackings, T_B, allowed during the fastNN search. Moreover, drop in isolated recognition accuracy is negligible for the EasyDigits, KinectT and upper6D datasets, while it is close to 1% for the digits6D and lower6D datasets when T_B is lower than 25% of the number of fastNN tree examples (Fig. 6.9).

Discussion While the above results are quite encouraging, one should keep in mind that they may vary for different random splits of digits6D users into the three sets (digitsTree, digitsR and digitsT). In our experiments, we explored only a few such splits,

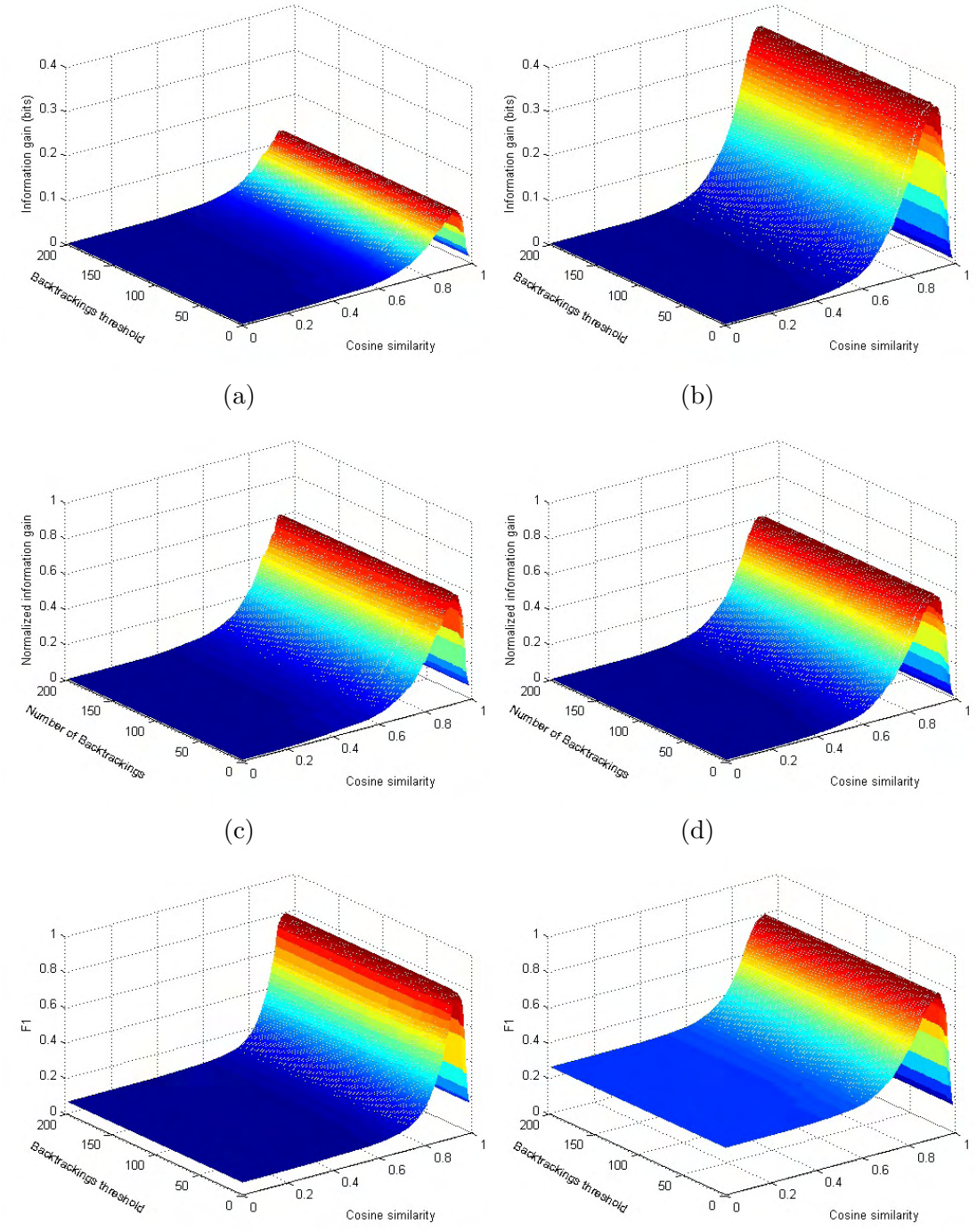
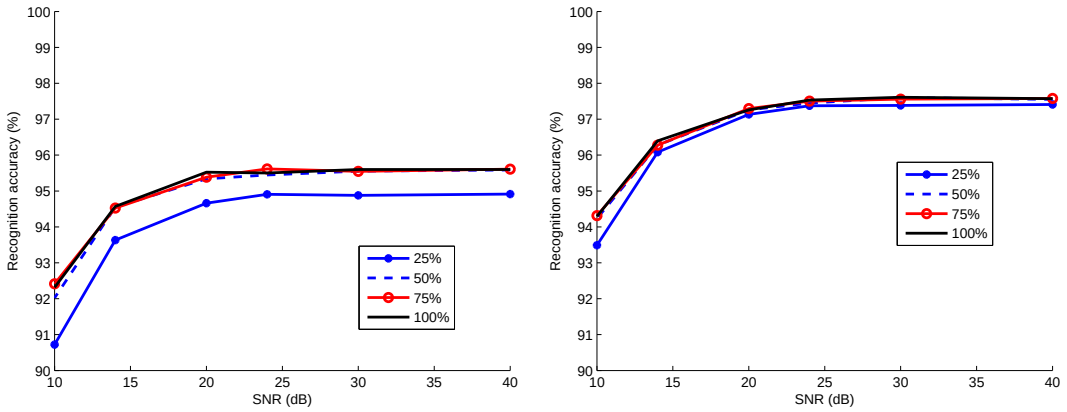


FIGURE 6.8: (a-b): Evaluation of gesture verification Rule1 by a system trained on 6D digits, based on the information gain they provide over the prior class probabilities. *Backtrackings threshold* indicates the number of backtrackings allowed during the fastNN search. The goal is to reject upper (a) and lower (b) letters in a system trained on digits. (c-d) Corresponding normalized information gains for the same evaluation. (e-f) F1 score for the same evaluation.

mainly due to the high computational cost imposed by DTW8. Although the results varied a little, on average they were close to those presented in this section, with DTW8 showing the best overall behaviour, followed by *Rule1* and *Rule2*. On the other hand, un–normalized DTW always presented the worst performance, reinforcing the importance of normalization.



(a) (b)

FIGURE 6.9: Isolated recognition accuracy (%) for noisy gestures of (a) lower and (b) upper–case letters, when a fixed number of backtrackings, T_B, is allowed during the fastNN search. T_B is shown as a percentage over the number of fastNN tree examples.

6.3.1.2 Separating digits from other mathematical symbols

Since the Kinect datasets include other mathematical symbols besides digits, we explored the case of separating digits from other mathematical symbols, considered as out–of–vocabulary gestures. The experimental setup is shown on Table 6.2. Although "KinectNonDigits" contains gestures completely different to digits, un–normalized DTW yielded very poor results. On the other hand, *Rule1* and DTW8 showed excellent performance ($EER = 1$), as depicted by the ROC curves in Fig. 6.10. Quite interestingly, *Rule2* showed $EER \approx 0.93$, although this is mainly a problem of Recall and not of Precision. Indeed, by choosing $T_B = 100$ (20% of the fastNN tree examples), *Rule2* achieved Precision–Recall $(1.0, 0.99)$ for the training and $(0.98, 0.93)$ for the testing set.

TABLE 6.2: Experimental setup for gesture verification on the Kinect datasets

Usage	**dataset**	**Number of examples**
fastNN tree	KinectR	500
Positives (Train)	KinectR	479
Negatives (Train)	KinectNonDigits	10
Positives (Test)	KinectT	54
Negatives (Test)	KinectNonDigits	40

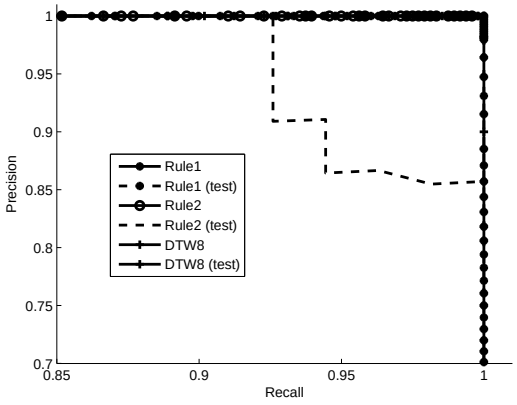


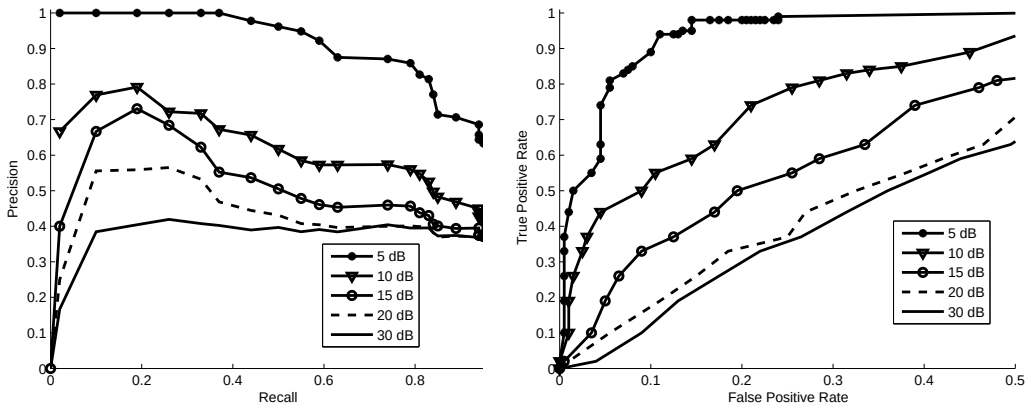
FIGURE 6.10: ROC curves for the various gesture verification methods on the Kinect dataset.

We also saw that DTW8 and *Rule1* could achieve information gains equal to the initial class entropies, i.e. $I(X;Y) = 0.144$ bits for the training and $I(X;Y) = 0.984$ bits for the testing set, thus reducing uncertainty to 0 bits. On the other hand, *Rule2* could achieve at most $I(X;Y) = 0.134$ bits for the training and $I(X;Y) = 0.778$ bits for the testing set. Although a significant amount of uncertainty still remains after applying *Rule2*, this is not a major drawback, since it can be eliminated through *Rule1*, provided that most of the positive examples are accepted by the system ($Recall \approx 1$). However, the above results may have been affected by the small size of the Kinect dataset.

6.3.2 Case 2: completely random gestures

In this case, we generated (x, y) values drawn from a Gaussian random process $\mathcal{N}(0, \sigma^2)$ and used them for testing the proposed system. Using 200 digits from the digits6D dataset to build the fastNN tree, we observed that *Rule2* could achieve perfect separation for $T_B \approx 30$, i.e. 15% of the number of fastNN tree examples. Indeed, random gestures caused an average value of 185 backtrackings with a standard deviation of 27, while positive examples required no more than 110 backtrackings ($\mu = 10, \sigma = 9$). Similar results were obtained for other datasets (EasyDigits and Kinect).

The above results prove that random trajectories, if no care is taken, may lead to increased complexity during tree search, causing a very large number of backtrackings (close to 90% of the tree examples). Early elimination of such gestures secures low complexity and high system reliability.



(a) (b)

FIGURE 6.11: ROC curves for rejection (using Rule1) of noisy digits6D gestures for different SNR values. (a) Standard Precision–Recall curves (b) True Positive Rate – False Positive Rate curves. Separation is easier for higher noise.

6.3.3 Case 3: noisy gestures

This case involves vocabulary gestures with some additive noise $\mathcal{N}(0, \sigma^2)$ and models devices with sensor noise or users with trembling hand. A desirable property for a recognition system would be to tolerate low noise levels and reject gestures with high noise. In our experiments, we split the digits6D dataset into three subsets (*digitsTree*, *digitsR*, *digitsT*), as in Sec. 6.3.1. We treated *digitsT* as an invalid dataset, by adding Gaussian noise at various SNR values (5 dB – 30 dB), while the objective was to check discrimination power of the proposed system between *digitsR* and *digitsT*. Our target was to verify increasing discrimination with increasing noise levels.

As we see in Fig. 6.11, discrimination ability of *Rule1* between positive (i.e. noise-free digits) and negative (i.e. noisy digits) examples is poor for low noise levels, since gestures still remain recognizable. However, when $SNR \leq 15dB$, the amount of noise is such that it renders them closer to random gestures than valid digits.

6.3.4 Relationship of number of fastNN backtrackings to probability of inlier

As described in Sec. 6.2.3.3, it is possible to associate a probability of inlier, P_0, to a query gesture, q, based on its Mahalanobis distance, $D^2(X, q)$, from a training set of valid examples, X. In this section, we explore the relation between this probability and the number of backtrackings, B, present during the fastNN search. As we show, outliers produce a higher number of backtrackings (B is higher for lower P_0), providing additional insight to the use of *Rule2* for gesture verification.

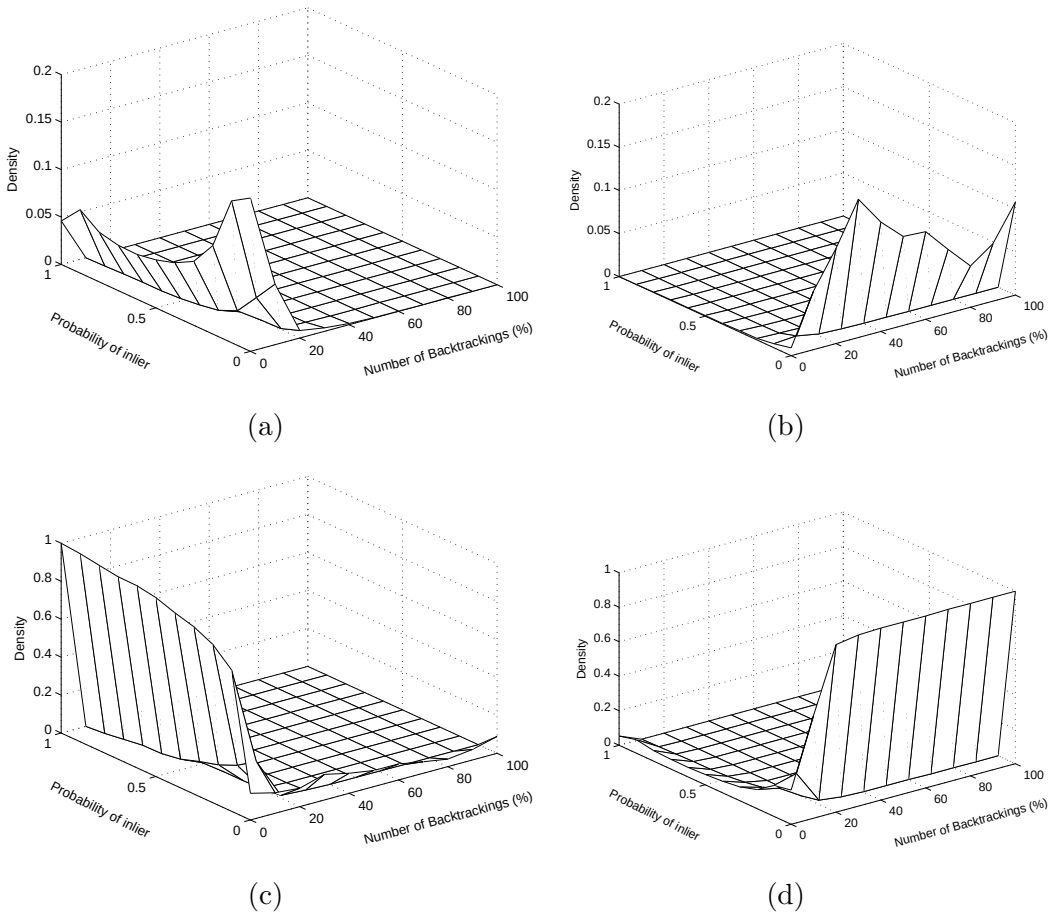


FIGURE 6.12: Joint distribution of inlier probability and fastNN backtrackings, (P_0, B), for (a) digits and (b) out–of–vocabulary gestures, on a system trained on the digits6D dataset. B is shown as a percentage over the number of fastNN tree examples. (c) Conditional distributions of fastNN backtrackings for fixed inlier probability values, $Prob\{B|P_0\}$ for a mix of digits and out–of–vocabulary gestures (at a ratio of 15 valid to 85 invalid gestures). (d) Conditional distributions of inlier probability values for fixed numbers of fastNN backtrackings, $Prob\{P_0|B\}$.

To this end, we repeated the experiments of Sec. 6.3.1 on the three 6D datasets (digits6D, upper6D, lower6D), in order to calculate the probability of inlier in addition to the number of backtrackings, thus collecting pairs (P_0, B) for the positive and negative examples. We split the digits6D dataset into two disjoint sets: a training set, containing 2 out of 5 users, used to build the fastNN tree (*digitsTree* set) and a testing set, containing 3 out of 5 users, used to simulate positive query vectors (*digitsRT* set). Additionally, we used the complete upper6D and lower6D datasets to simulate negative query vectors. Please note that we considered all 5 user combinations for the digits6D dataset, resulting in $5! = 120$ splits, to collect more robust statistics.

Fig. 6.12 (a), (b) show the joint distributions of (P_0, B) pairs for valid and invalid gestures, correspondingly. We observe that invalid gestures exhibit lower inlier probability, P_0, while they cause a larger number of backtrackings, B, on average. One can trivially

verify that query vectors from the training set (digitsTree) result in cosine similarity values almost equal to 1, number of backtrackings almost 0 and probability of inlier very close to 1. On the other hand, valid gestures from other users result in relatively high P_0 values ($P_0 \in [0.2, 0.5]$) with relatively low number of backtrackings ($B < 20\%$). For comparison, most of the negative examples show $P_0 < 0.2$ and $B > 20\%$. P_0 values between 0.5 and 1 were hard to be obtained in a user independent mode (digitsRT set), since different users have intrinsically different distributions of gesture vectors. In order to generate such values, we introduced low levels of additive Gaussian noise to the training vectors (digitsTree) and used them as query vectors; in our figures, values higher than 0.7 are in their majority due to noisy versions of the fastNN training examples.

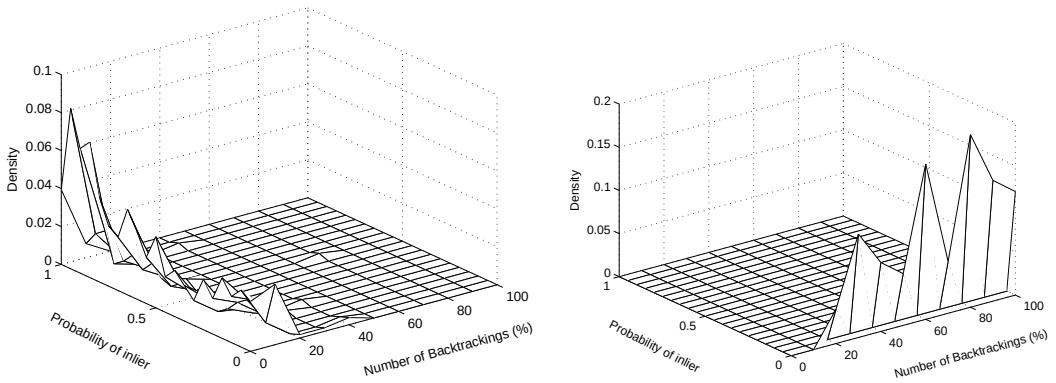
Fig. 6.12–c shows the conditional distributions of fastNN backtrackings, B, for fixed inlier probability values, P_0. We notice that the number of backtrackings, B, is less than 20% of the total number of tree examples with very high probability when $P_0 > 0.2$, corresponding to valid query vectors. On the other hand, lower values of P_0, corresponding to invalid query vectors, result in an almost random number of backtrackings, with a bias towards larger ($B > 20$) values. Similarly, P_0 is almost fixed to 0 for $B > 20\%$, while it varies for lower B (Fig. 6.12–d). In other words, query vectors that require more than 20% of the total number of tree examples in backtrackings are almost certainly out–of–vocabulary gestures.

We also performed similar experiments on the Kinect dataset, using the experimental setup of Table 6.2. As we observe in Fig. 6.13, results are quite similar to those of Fig. 6.12, although a different dataset is used.

Finally, we performed a similar experiment on the upper6D dataset, considering 13 letters as positive and 13 as negative classes. We used 5 (out of 13) users to build the fastNN tree, and 2 distinct users for each of the other four subsets (positives/negatives $\times$ validation/testing combinations). We repeated this experiment 100 times, each with different random splits of users and classes. Fig. 6.14 shows the resulting distributions, where we observe the same pattern for positive and negative examples.

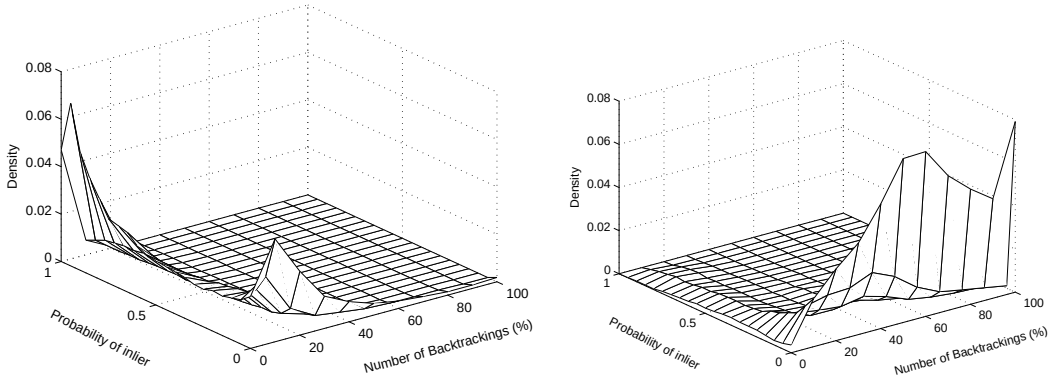
6.3.5 Relating cosine similarity score and number of fastNN backtrackings

As already suggested by the effectiveness of Rule3 (Fig. 6.6), cosine similarity score, P is correlated with the number of fastNN backtrackings, B. Indeed, as Fig. 6.15 shows for a system trained on 6D digits and tested on 6D digits and letters, positive examples exhibit higher values of P compared to negative examples, as well as lower number of backtrackings, B. These figures agree with the results of Fig. 6.6 for *Rule3*.



(a) (b)

FIGURE 6.13: Joint distribution of inlier probability and fastNN backtrackings, (P_0, B), for (a) digits and (b) out–of–vocabulary gestures, on a system trained on the KinectR digits dataset. B is shown as a percentage over the number of fastNN tree examples.



(a) (b)

FIGURE 6.14: Joint distribution of inlier probability and fastNN backtrackings, (P_0, B), for (a) valid and (b) out–of–vocabulary gestures, on a system trained on the upper6D dataset, for 100 random trials. B is shown as a percentage over the number of fastNN tree examples.

To better explore the relationship between B and P, we performed several experiments, varying the number of training examples and data dimensionality. Fig. 6.16–a shows collected pairs (P, B), indicating that B can be coarsely modelled as a logistic function of P, i.e.

$$B = \frac{1}{1 + e^{-\theta_0 - \theta_1 P}} \tag{6.9}$$

Indeed, such a model makes more sense than a linear one, since B converges to 100% for $P = 0$ and 0% for $P = 1$. As Fig. 6.16–b shows, this model achieves 9% error on average between predicted and actual values of B on average. Similar error values were measured in different pairs, which were used as validation and testing sets, thus

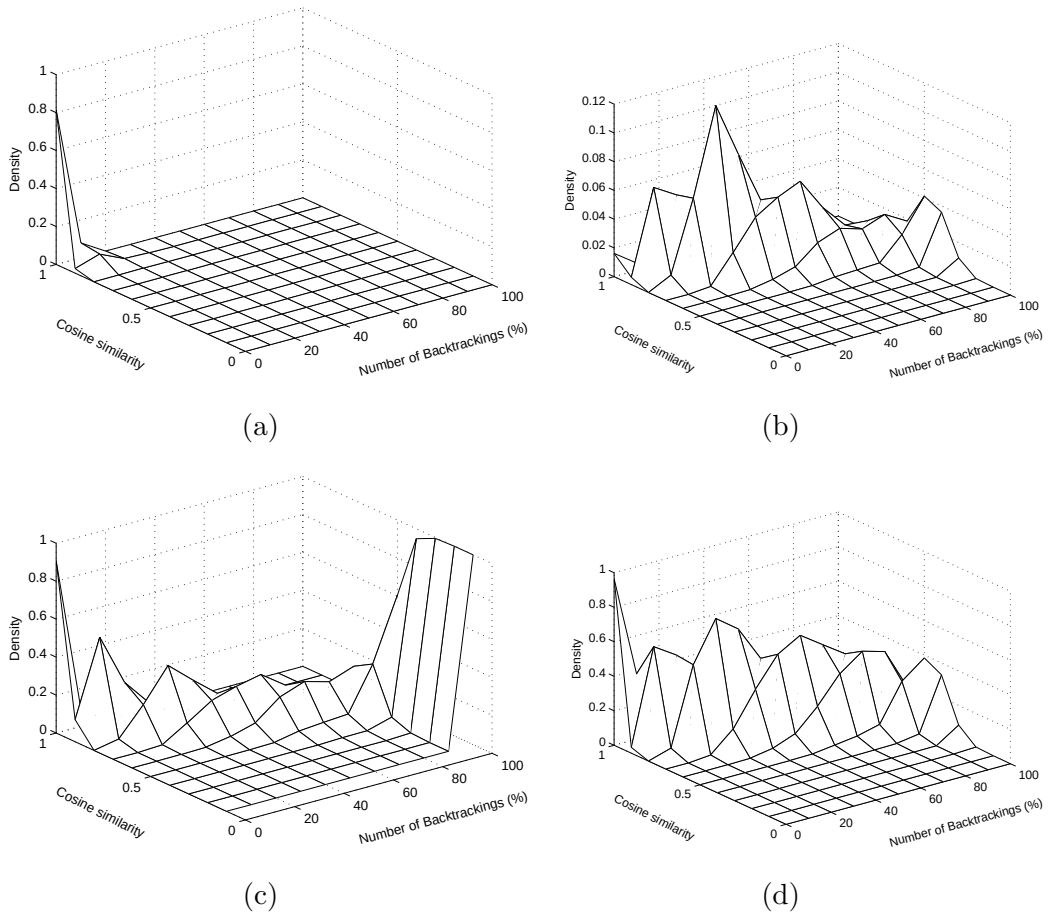


FIGURE 6.15: Joint distribution of cosine similarity and fastNN backtrackings, (P, B), for (a) digits and (b) out–of–vocabulary gestures, on a system trained on the digits6D dataset. B is shown as a percentage over the number of fastNN tree examples. (c) Conditional distributions of fastNN backtrackings for fixed cosine similarity values, $Prob\{B|P\}$ for both digits and out–of–vocabulary gestures. (d) Conditional distributions of cosine similarity values for fixed numbers of fastNN backtrackings, $Prob\{P|B\}$.

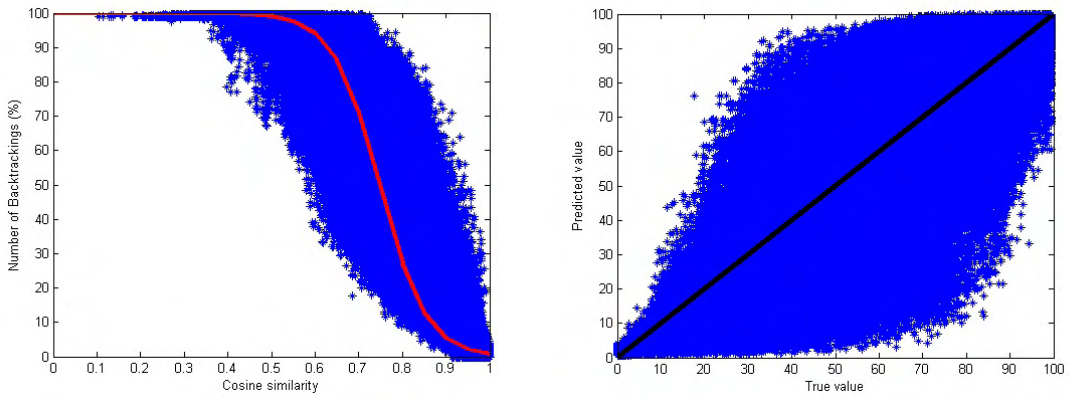
indicating a case of high bias due to the use of just one feature (cosine similarity score). Parameters θ_i were found to be $\theta_0 = 11.5, \theta_1 = -15.3$.

The error decreased to 8% after adding the inlier probability, P_0, as an additional feature and refitting a logistic function with both features, i.e.

$$B = \frac{1}{1 + e^{-\theta_0 - \theta_1 P - \theta_2 P_0}} \tag{6.10}$$

where $\theta_0 = 8.5, \theta_1 = -6.7, \theta_2 = -11.1$.

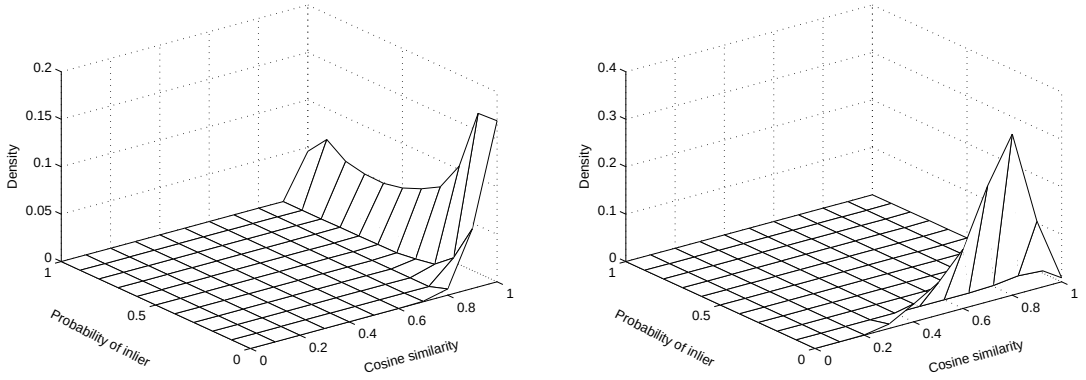
This small improvement is not surprising, since P and P_0 are correlated, as suggested by Fig. 6.17, although P_0 seems to take lower values than P for both positive and negative examples. Additionally, P_0 varies mostly for positive examples, while P varies mostly



(a) (b)

FIGURE 6.16: Modelling the number of backtrackings, B, as a logistic function of the cosine similarity score, in a system trained on 6D digits and tested with 6D digits and letters.

for negative examples. Seeking richer models and more expressive features is included in our goals of future work.



(a) (b)

FIGURE 6.17: Joint distribution of cosine similarity and probability of inlier, (P, B), for (a) digits and (b) out–of–vocabulary gestures, on a system trained on the digits6D dataset.

6.3.6 Effect of dimensionality

Finally, we questioned the relationship between data dimensionality and gesture verification performance. To this end, we repeated the experiment of Sec. 6.3.1, varying the number of resampled points, N.

Fig. 6.18 shows the ROC curves produced after applying Rule1 (cosine similarity score) and Rule2 (number of backtrackings) on the testing sets (*digitsT* versus lower–case letters) when 4, 8 and 16 points are kept after resampling. Clearly, $N = 4$ is not a

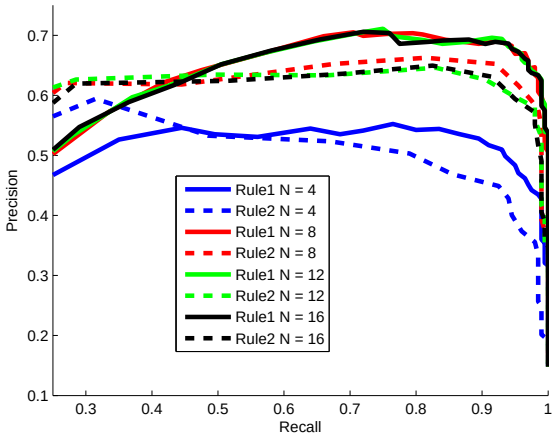
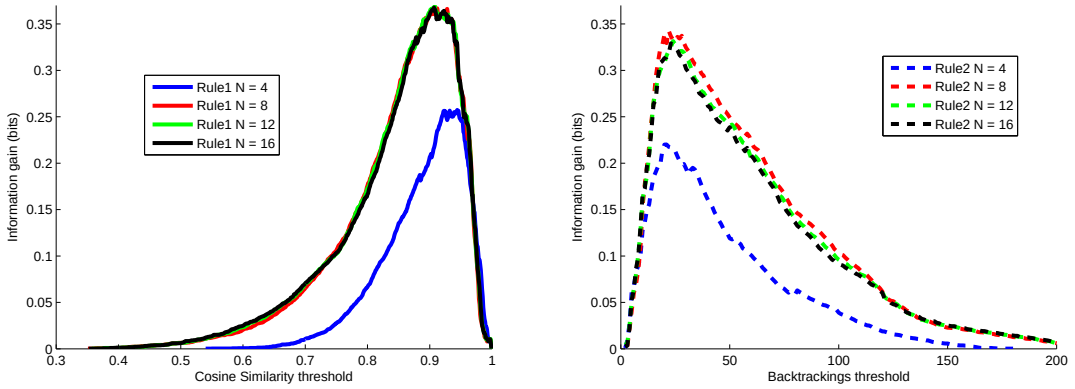


FIGURE 6.18: ROC curves with varying number of resampling points, N. Verification performances seem to agree with recognition accuracies, in Fig. 5.2.

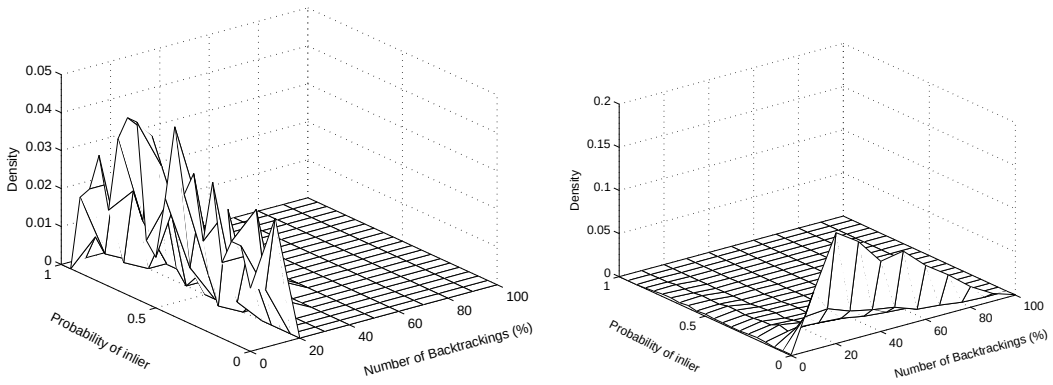
good choice, as it also presents reduced recognition performance (Fig. 5.2). On the other hand, $N = 8$, $N = 12$ and $N = 16$ show rather similar verification performance, agreeing with their similar recognition capabilities.



(a) (b)

FIGURE 6.19: Information gains of the two MCS rules with varying number of resampling points, N, and threshold values. *Rule1* (a) performs better than *Rule2* (b), offering 0.0385 more bits of information for $N \geq 8$, confirming the result of Fig. 6.18.

The above results can be verified by computing information gains of the two rules at various values of N, as shown in Fig. 6.19. A more intuitive verification arrives by inspecting the distributions of inlier probability–fastNN backtrackings pairs. Indeed, when $N = 4$, the boundaries between positive and negative examples become less clear, as shown in Fig. 6.20.



(a) (b)

FIGURE 6.20: Distribution of inlier probability–fastNN backtrackings pairs, (P_0, B), for (a) digits and (b) out–of–vocabulary gestures, on a system trained on the digits6D dataset, when only 4 trajectory points are kept after resampling. B is shown as a percentage over the number of fastNN tree examples. One can compare to the *clearer* plots when 8 points are used, in Fig. 6.12.

6.4 Minimum backtrackings classification

Since the original application of fastNN search was vector quantization for image coding purposes, the number of *backtrackings*, B, was interpreted only as a rough measure of computational efficiency. In our application, where the goal is classification, B could also indicate the difficulty faced by fastNN to classify a specific query gesture. As described in Sec. 5.1.2, fastNN arranges the training examples in a binary tree, during its initialization phase. Our main assumption is that as long as a query vector, q, is similar to one of the training examples, B (and the search time) will be relatively small. On the other hand, if q is dissimilar to the training examples, then B will be quite large.

Based on the above hypothesis, we train C fastNN trees, one for each of the C gesture categories. Given a query vector, q, we perform C fastNN searches, and measure the corresponding cosine similarity–backtracking pairs, $(p_c, B_c), c = 1, \ldots, C$. To deal with skewed classes, we compute the normalized number of backtrackings, $b_c = \frac{B_c}{M_c}$, where M_c denotes the number of training examples in class c. We then locate the class presenting the minimum b_c and assign its label to q. We denote this method as *MBC*.

An alternative choice would be to combine p_c, b_c and compute a common score, $PB_c = {p_c}^{\alpha} \cdot {b_c}^{\beta}$, or equivalently, $PB_c = \alpha \log p_c + \beta \log b_c$. In this case, the query vector is assigned to the class showing the maximum PB_c. We denote this method as *MBC–MCS*.

Table 6.3 shows recognition results on the four gesture datasets, for noiseless gestures and the maximum possible number of users. Quite interestingly, MBC achieves good

TABLE 6.3: Recognition accuracy (%) using minimum backtrackings classification

Dataset/Method	MBC	MCS	MBC–MCS
EasyDigits	89.3	97.3	97.3
digits6D	84.5	98.7	99.2
lower6D	65.9	95.5	96.2
upper6D	71.5	97.6	97.6

recognition accuracy for the digit datasets ($> 84\%$) and moderate accuracy on the letter datasets ($> 65\%$). Moreover, recognition accuracy drops when digits contain noise, as highly noisy gestures are treated as invalid gestures. As we see in Fig. 6.21, when SNR drops to extremely low values ($SNR < 10$), recognition accuracy becomes 10%, i.e. equal to the accuracy of random prediction.

Overall, MBC is neither the best classifier, nor the most suitable for gesture recognition. However, the above results hopefully provide additional insight to the link between computational complexity and gesture recognition.

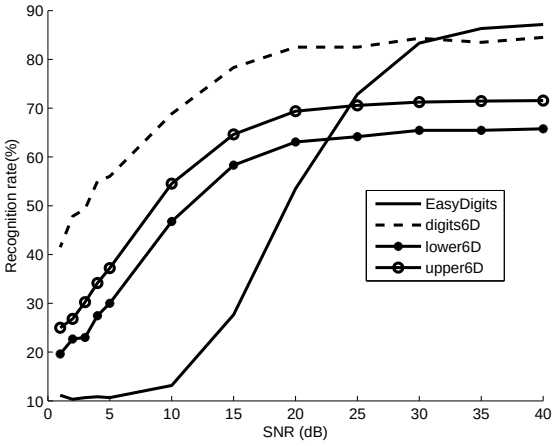


FIGURE 6.21: Recognition accuracy of the MBC classifier for noisy gestures. As expected, accuracy decreases with noise.

6.5 Discussion

In this chapter, we presented our approach for gesture verification and rejection of invalid gesture instances, based on thresholding the cosine similarity score and the number of fastNN backtrackings required to classify an unknown gesture vector. Our approach can effectively reject three main types of invalid gestures: *out–of–vocabulary* gestures, *noisy* gestures and *random* movements.

We additionally showed that there exists a relationship between the number of backtrackings, cosine similarity score and the probability of inlier. This property leads to high computational efficiency, as we may constrain the search time up to a certain number of backtrackings, rejecting all other inputs as invalid gestures.

Finally, we showed a weak relationship between computational efficiency and recognition accuracy, which may prove quite promising in the future.

Chapter 7

Gesture spotting

Contents

7.1 Introduction

The final component of our approach involves gesture spotting on continuous hand coordinate streams, which poses the joint problem of detecting time boundaries and recognizing gestures simultaneously. As already described in sections 1.3 and 2.2, spotting methods may either try to first detect the time boundaries of the performed gestures

and then apply standard isolated recognition techniques (*direct approach*), or form joint hypotheses about both boundaries and categories of gestures (*indirect approach*). Our approach is indirect, applying isolated recognition on multiple sliding windows, collecting groups of conflicting overlapping gesture candidates and finally choosing the most likely candidate.

Based on the results of Chapter 5, we know that Maximum Cosine Similarity (MCS) recognizes the correct gesture category with probability $> 96\%$ for noiseless gestures, provided that the gesture segmentation in time is accurate. We also know that a significant amount of invalid gestures may be rejected by gesture verification rules, as described in Chapter 6. Remaining invalid gestures, along with subgestures, i.e. meaningful parts of longer gestures, and false positive gestures due to time window misalignment, form groups of conflicting, overlapping gesture candidates at each time frame, out of which only one corresponds to the ground–truth gesture. We refer to the process of evaluating overlapping candidates, such that only one is selected and reported by the spotting module, as *conflict resolution*.

Conflict resolution is a challenging task because the trivial rule of choosing the candidate of highest cosine similarity score (or lowest distance in Dynamic Time Warping [2]) cannot produce high spotting results. Typically, some subgesture modelling is required, which takes into account the relationships between conflicting gesture candidates. To this end, we propose a novel conflict resolution approach and explore its variations and behaviour, based on specific parameters and gesturing scenarios.

In short, the major contributions of this chapter are:

- Proposing a novel method to perform conflict resolution, combining both the cosine similarity score (measuring the quality of recognition) and the time duration of gesture candidates.
- Introducing the usage of time boundaries when learning subgesture relationships, which improves spotting results.
- Designing a special version of the Weighted Edit Distance [123] for recognition of vocabulary words.
- Exploring various aspects of gesture spotting, such as time delay in announcing spotting results, connectivity between gestures and order of gesture appearances.

In all cases, we conduct thorough experiments on real and synthetic gesture datasets, as well as on real English literature texts.

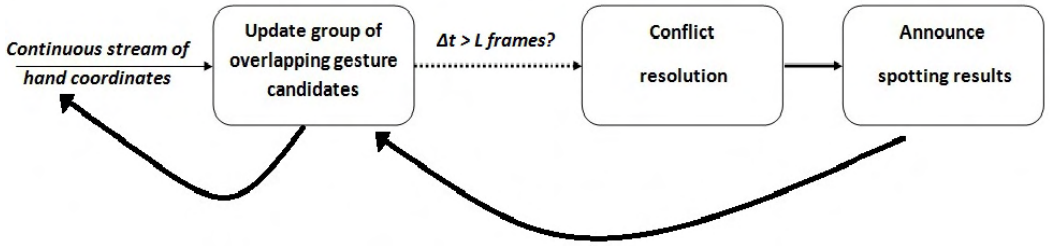


FIGURE 7.1: General structure of our gesture spotting approach. A group of overlapping gesture candidates is formed until there is no further update or a certain maximum time has elapsed from the earliest detected gesture. At that point, conflict resolution takes place, leading to a possible announcement of spotting results.

7.2 Our approach

In this section, we describe our approach in detail. A block diagram is shown in Fig. 7.1.

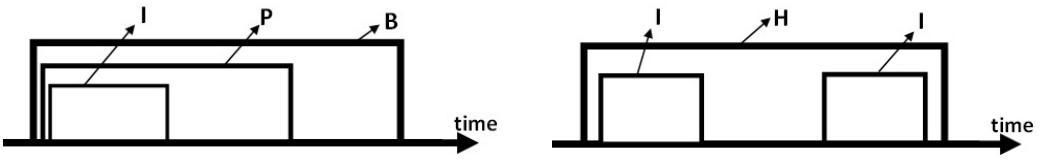
7.2.1 Spotting single gestures

7.2.1.1 Forming groups of overlapping gestures

Assuming that a continuous stream of hand coordinates, (x_t, y_t) for each time frame, t, is available, the first step of our approach is to form groups of conflicting overlapping gesture candidates, which will be further processed in the next step (*conflict resolution*), such that only one candidate survives.

Specifically, we consider W windows of various time durations, centered at each frame, t, and apply fastNN–MCS–based isolated gesture recognition, as presented in Chapter 5. Each window produces one gesture candidate, g, of cosine similarity value P_g, number of fastNN backtrackings, B_g, and time duration, w_g. We then apply gesture verification, as described in Chapter 6, using *Rule3*, with some properly chosen thresholds T_B and T_P, to remove those candidates that are clearly invalid (low P_g and/or high B_g).

Subsequently, we combine P_g and w_g, favoring those gestures that are similar to training examples (higher P_g), but also those that are longer than others (higher w_g). This is done in order to avoid the *subgesture* problem, i.e. the fact that certain gestures contain other gestures in their shape. Classic examples are the letters "I, P" that appear while drawing the letter "B" (Fig. 7.2–a). Intuitively, we would like to favor longer gestures over shorter ones, when they have comparable cosine similarity metrics.



(a) (b)

FIGURE 7.2: Two typical cases of the subgesture problem. (a) Letters "I" and "P" also appear while gesturing "B". (b) Gesturing letter "H" involves two appearances of the letter "I". Please note that the order of appearance is different (and thus discriminative).

To this end, we define the combined score of a gesture g, having time duration w_g and cosine similarity P_g, as:

$$PT_g = (w_g)^\alpha \times (P_g)^\beta \tag{7.1}$$

where α, β are shape parameters that can be established through cross–validation. An alternative way of viewing the above score is as a linear combination of the logarithms of T_g and P_g. For our experiments, we chose $\alpha = 1$ and $\beta = 32$ through cross–validation on a training set (Sec. 7.3.1.2).

Surviving gesture candidates (g, PT_g) are added in a candidate list. If two entries belong to the same gesture category, we keep the one with higher cosine similarity score. Additionally, we keep note of the candidate with the earliest starting time t_0, as well as of the candidate with the highest combined score, PT_M. L frames after t_0 we find the group of overlapping candidates in $[t_0, t_0 + L]$, sort them in descending order based on their PT scores and keep the top K candidates that exceed a certain percentage of the highest combine score, $\lambda \times PT_M$, $g_i, i = 1, \ldots, K$, as the final conflict group. Using a candidate list implies some time delay in the final reporting of spotted gestures, problem common to all indirect gesture spotting systems [2].

Parameter L was chosen to guarantee inclusion of the maximum duration training example. Parameters λ and K were selected in order to provide a good compromise between recognition accuracy and computational complexity. Since we found $\lambda = 0.45$ and $K = 4$ to yield very good results on training sets, we chose such values for experiments on all other datasets.

7.2.1.2 Conflict resolution

Conflict resolution is performed by taking advantage of the subgesture relationships appearing during gesture formation. Specifically, we noticed that different gestures involve different subgestures, which appear at different relative time positions. For example, in Fig. 7.2–a, gesturing letter "B" produces a candidate gesture "I" in the first one–third of the total time duration, a candidate "P" in the last 30% and a candidate

"B" in the last 10%. Similarly, gesturing letter "H" produces a candidate "I" in the first 33% and a second "I" in the last 33% of the total time duration.

To learn such subgesture relationships, we ran the spotting process on training sets and collected statistics, namely conditional probabilities, regarding class labels and their order of appearance in the final conflict group, as well as their starting, t_s^i and ending times, t_e^i, relative to the minimum and maximum group times. To reduce table size, we uniformly quantized time values in $N_b = 5$ bins, with respect to the duration of the longest gesture candidate. The underlying Bayesian network is shown in Fig. 7.3.

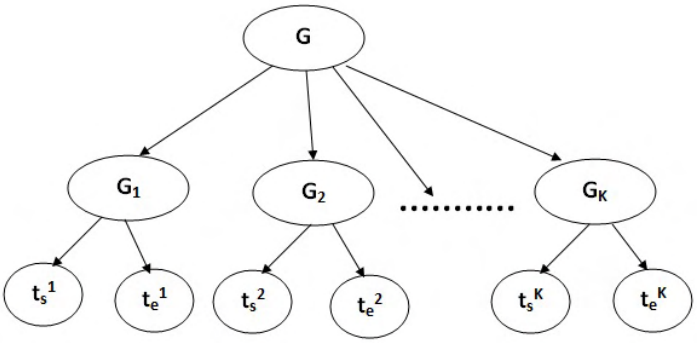


FIGURE 7.3: The Bayesian network which models the subgesture relationships between gesture candidates.

During testing, we choose the candidate of maximum likelihood $L(g_i)$ as the final spotted gesture:

$$L(g_i) = Prob[g_1, t_s^1, t_e^1, \ldots, g_k, t_s^k, t_e^k, g_i] \tag{7.2}$$

$$= Prob[g_i] \cdot Prob[g_1, t_s^1, t_e^1, \ldots, g_k, t_s^k, t_e^k | g_i] \tag{7.3}$$

$$= Prob[g_i] \cdot \prod_{k=1}^{K} Prob[g_k, t_s^k, t_e^k | g_i, g_1, t_s^1, t_e^1, \ldots, g_{k-1}, t_s^{k-1}, t_e^{k-1}] \tag{7.4}$$

$$= Prob[g_i] \cdot \prod_{k=1}^{K} Prob[g_k, t_s^k, t_e^k | g_i] \tag{7.5}$$

$$= Prob[g_i] \cdot \prod_{k=1}^{K} Prob[g_k | g_i] \cdot Prob[t_s^k | g_i, g_k] \cdot Prob[t_e^k | g_i, g_k] \tag{7.6}$$

where we make the simplifying assumption that, given the correct gesture category, occurrences of gesture candidates are mutually independent and that ending time, t_e^k, is independent of the corresponding starting time, t_s^k.

Assuming a uniform prior distribution over the vocabulary gestures, we get:

$$L(g_i) = \prod_{k=1}^{K} Prob[g_k | g_i] \cdot Prob[t_s^k | g_i, g_k] \cdot Prob[t_e^k | g_i, g_k] \tag{7.7}$$

In practice, one uses the log–likelihood, to avoid numerical calculation underflows:

$$\ell(g_i) = \sum_{k=1}^{K} \log Prob[g_k|g_i] + \log Prob[t_s^k|g_i, g_k] + \log Prob[t_e^k|g_i, g_k] \tag{7.8}$$

7.2.1.3 Conflict resolution variations

While our subgesture relationships model is quite rich, in the sense that it models both the gesture categories and their starting and ending times, it involves many probabilities that have to be estimated from training data. Indeed, for a vocabulary of C gesture categories, our complete subgesture model needs to estimate $M_{class} = C \cdot (C-1)$ probability values for $Prob[g_k|g_i]$, $M_{start} = C \cdot (C-1) \cdot N_b$ values for $Prob[t_s^k|g_i, g_k]$ and $M_{end} = C \cdot (C-1) \cdot N_b$ values for $Prob[t_e^k|g_i, g_k]$.

For the specific cases of digits and letters, we need to estimate $M_{class} = 90$, $M_{start} = 450$, $M_{end} = 450$ probability values for the 10 digits and $M_{class} = 650$, $M_{start} = 3250$, $M_{end} = 3250$ values for the 26 letters. While these numbers are quite large, especially for the case of letters, the probability spaces are expected to be quite sparse, since a lot of subgesture relationships may never appear in practice. For example, it is highly improbable –or even impossible– to detect a candidate of letter "A" when actually writing letter "Z". Similarly, starting and ending relative times present sparsity, which decreases the actual number of parameters. Nevertheless, we acknowledge that the number of training examples could be a possible issue in subgesture relationship learning, which should not be overlooked by a practical application system.

Based on the above observations, we considered the following variations on the subgesture relationships model:

- *Complete subgesture model* : complete model of subgesture relations, modelling gesture categories as well as starting and ending time bins.
- *Class–only model* : reduced model, modelling only gesture dependencies, by ignoring starting and ending times.
- *Starting–time model* : reduced model, modelling gesture categories and starting times, by ignoring ending times.
- *max PT score* : no model at all, simple conflict resolution by reporting the gesture candidate with the highest PT score.

Please note that *Class–only model* is different from the method described by Alon et al.[2]. While they also measured frequencies of subgesture occurrences, they never actually used them. Instead, their conflict resolution method involved hard subgesture relationship rules, stating that a subgesture should always be replaced by a supergesture, when applicable. Thus, our work builds on the method of Alon et al., extending it such that it handles subgesture relationships in a probabilistic fashion. Furthermore, we introduce modelling relative time boundaries between subgestures and supergestures (*starting–time model* and *complete subgesture model*).

7.2.2 Recognizing gesture words

While so far we considered inputs of separate digits and letters, it is also possible for the user to gesture words that belong to a predefined vocabulary. In this case, we assume that word boundaries can be easily detected by some *direct* scheme, through detection of abrupt motion cues and pauses. We then use our method for gesture spotting to detect single letters and perform word recognition by minimizing the Weighted Edit Distance [123] between the query and vocabulary words, as described below. Although it would be possible to avoid spotting altogether and recognize the complete word trajectory as a single entity through standard isolated recognition, such method would be highly prone to errors, due to different protocols of connecting word letters, as we will show in Sec. 7.4.

7.2.2.1 Weighted Edit Distance for word gesture recognition

Assuming two sequences of letters (*strings*), $X = \{x_1, x_2, \ldots, x_M\}$ and $Y = \{y_1, y_2, \ldots, y_N\}$, the Edit Distance is defined as the minimum number of editing operations (insertions, deletions and substitutions) required to transform sequence X to Y [124]. Computing the Edit Distance, $D(X, Y)$, is based on Dynamic Programming and resembles the computation process of Dynamic Time Warping (DTW) [112].

Specifically, let $D(i, j)$ denote the Weighted Edit distance between the first i and j letters of sequences X and Y respectively. Then, $D(i, j)$ can be computed recursively as:

$$D(i,j) = \min \begin{cases} D(i-1,j) + del[x_i] \\ D(i,j-1) + ins[y_j] \\ D(i-1,j-1) + sub[x_i, y_j] \end{cases} \tag{7.9}$$

where $del[x_i]$ defines the cost of deleting x_i, $ins[y_j]$ the cost of inserting y_j and $sub[x_i, y_j]$ the cost of substituting x_i with y_j. To initialize the process, we assume $D(i,0) = del[x_i], D(0,j) = ins[x_j], \forall i,j$.

A simplified version, called Levenshtein Distance or simply Edit Distance [124], considers fixed edit costs as $del[x_i] = 1$, $ins[y_j] = 1$ and

$$sub[x_i, y_j] = \begin{cases} 0, & \text{if } x_i = y_j \\ 2, & \text{if } x_i \neq y_j \end{cases} \tag{7.10}$$

However, in the general case, these costs vary, depending on the application. For example, text processors relate the substitution cost, $sub[x, y]$, to the probability of mistyping letter y instead of x, which depends on the relative positions of letters in keyboards. In our approach, we use a training set of gesture sequences to measure the following probabilities:

- $Prob\{y|x\}$: probability of recognizing gesture y while x was performed (*substitution error*)
- $Prob\{y|\emptyset\}$: probability of recognizing gesture y while no gesture was performed (*insertion error*)
- $Prob\{\emptyset|x\}$: probability of recognizing no gesture while gesture x was performed (*deletion error*)
- $Prob\{\emptyset|\emptyset\}$: probability of recognizing no gesture while no gesture was performed (*neutral case*)

Usage of $Prob\{\emptyset|\emptyset\}$ is necessary to better normalize the probability distribution $Prob\{y|\emptyset\}$. We estimated it as:

$$Prob\{\emptyset|\emptyset\} = \frac{\textit{(\# of intervals)} - \textit{(\# of insertions)}}{\textit{(\# of intervals)}} \tag{7.11}$$

where C denotes the number of gestures in each training sequence and $\textit{(\# of intervals)} = (C-1) \cdot \textit{(\# of sequences)}$ the total number of intervals between consecutive gestures.

Finally, we define the edit costs as follows:

- $sub[x_i, y_j] = -\log Prob\{y_j|x_i\}$
- $ins[y_j] = -\log Prob\{y|\emptyset\}$
- $del[x_i] = -\log Prob\{\emptyset|x_i\}$

7.3 Experimental results for single gesture spotting

In this section, we present our experimental results for single[1] gesture spotting on continuous hand coordinate streams.

7.3.1 Experimental setup

7.3.1.1 Datasets

GreenDigits, EasyDigits [2], KinectStreamOrdered and KinectStreamRepeat [10] datasets already contain continuous gesture sequences of digits and we use them directly in our gesture spotting experiments. However, the 6D datasets contain only isolated gestures. Thus, we created artificially continuous sequences by connecting gestures with noisy straight lines of random time duration d frames ($d \in [70, 100]$, as measured on EasyDigits). Each sequence contains one instance from each gesture category, i.e. 10 for digits and 26 for letters. We considered two cases for the relative positions of consecutive gestures:

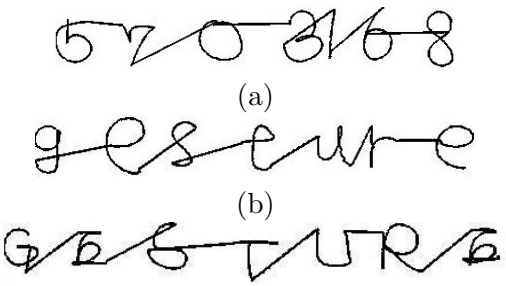
- *Case I* – user writes new gestures over previous gestures ("overwriting")
- *Case II* - user writes new gestures consecutively ("writing towards infinity")

Case I is adopted by the original EasyDigits and KinectStream datasets and models natural space limitations – i.e. the fact that there is a limited camera view angle towards the user. Case II (Fig. 7.4) models writing on paper and may occur in practice when the user writes two or more consecutive gestures side by side in front of the camera, before overwriting previously written gestures. It also appears in a completely different problem, when a users writes text with a digital pen on a tablet touch–screen[2]. Since a real–life application may involve a combination of the two writing cases, evaluating performance on both cases provides some lower and upper bounds on future system performance.

Digits in the EasyDigits and KinectStreamOrdered datasets appear in arithmetic order, while the KinectStreamRepeat dataset shows repetitions of similar digits. Thus, performance of previous spotting approaches [2, 10] on sequences of randomly appearing gestures is untested. For this reason, we also evaluated our method on *mixed* datasets,

[1] The term "single" appears to distinguish this process from "word gesture spotting", which involves recognizing complete words and not just characters. We explore word gesture spotting in Sec. 7.4.

[2] In theory, the presented approach could also be applied to recognize digits and letters written by digital pens on touch screens.



(a)

(b)

(c)

FIGURE 7.4: Three artificially created sequences, by connecting isolated gestures with straight line segments. (a) Digits6D: "5703168". (b) Lower6D: "gesture". (c) Upper6D: "GESTURE".

which were created synthetically by combining isolated gestures concatenated in random order[3].

During training, we collected subgesture statistics using the mixed datasets, in a user–independent mode. Please note that the mixed datasets used during testing were not the same as those used for training, in order to avoid overfitting. For the original EasyDigits dataset, we used statistics from the GreenDigits dataset. For the KinectStreamOrdered dataset, we learned statistics from KinectStreamRepeat. Finally, we split the Kinect-StreamRepeat dataset into two disjoint subsets, learned a subgesture model separately from each of them and applied it on the other subset. This way, spotting results remain unbiased, since training and testing sets are disjoint.

While we use a significant number of training sequences per user, as shown in Table 7.1, such number may not be enough in certain cases. Specifically, our main concern is about letters, where the number of probability distribution bins for starting and ending gesture frames, M_{start}, M_{end}, is extremely large ($M_{start} = 3250$). On the other hand, conditional probability bins for gesture categories involve a significantly reduced number ($M_{class} = 650$), and thus they may be learned more effectively. For such reasons, we evaluate all four conflict resolution methods (Sec. 7.2.1.3), i.e. *complete subgesture model*, *class–only model*, *starting–time model* and *max PT score*.

TABLE 7.1: Number of training sequences and model parameters for each dataset

Dataset	**Training sequences**	M_{class}	M_{start}
EasyDigits	2700	90	450
Digits6D	1500	90	450
Lower6D	1200	650	3250
Upper6D	3600	650	3250

[3]We plan to make these datasets available to researchers for further testing.

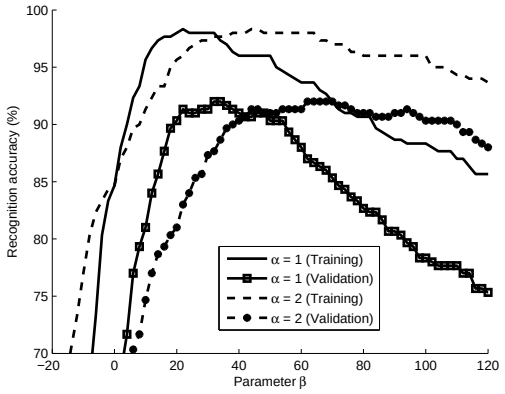


FIGURE 7.5: Recognition accuracy as a function of parameters α, β. Training was performed on GreenDigits and validation on EasyDigits.

7.3.1.2 Choosing parameters

During testing, a gesture is rejected based on *Rule3* (Sec. 6.2.1), i.e. if its cosine similarity is less than T_P and its number of backtrackings greater than T_B. To minimize false negatives, we chose conservative values, i.e. $T_P = 0.9$ and $T_B = 15 - 20\%$ of the (positive) training examples in the fastNN tree. The window lengths, W_i, were chosen based on the ranges of time durations in the training sets, with a step of 10 or 20 frames. During testing, we always considered a leave–one–user–out setup.

As discussed earlier in the analysis section 7.2.1.1, shape parameters α, β used in calculating each gesture's PT score, were given values $\alpha = 1$ and $\beta = 32$, by using GreenDigits for training and EasyDigits for validation – we used the original continuous datasets of [2] and not the artificial ones. Although we obtained comparable results for $\beta \in [20, 40]$ on both GreenDigits and EasyDigits (Fig.7.5), we finally chose $\beta = 32$, since computation of x^{32} can be efficiently implemented using 5 multiplications, as $x^{32} = x^{2^5}$ [4].

Values for parameters α, β, as well as those for $\lambda = 45\%$ (percentage of maximum PT score used to retain gesture candidates), $K = 4$ (number of top candidates) and $N = 5$ (number of time bins to store subgesture statistics) were kept fixed for all datasets. Parameter L (time window length for group of conflicting gestures) is set equal to 120% of the maximum duration of a gesture in a dataset, to guarantee inclusion of most probable gestures and time-windows (Tables 7.4 – 7.5).

[4] While this property is not crucial in floating–point arithmetic, it becomes valuable in fixed–point implementations.

7.3.1.3 Performance evaluation

Following [125], we measured Recognition accuracy (*Rec.*) and Reliability (*Rel.*), defined as:

$$Rec. = \frac{\#\ of\ correctly\ recognized\ gestures}{\#\ of\ input\ gestures} \tag{7.12}$$

$$Rel. = \frac{\#\ of\ correctly\ recognized\ gestures}{(\#\ of\ input\ gestures) + (\#\ of\ insertion\ errors)} \tag{7.13}$$

Similar to [2], we consider a detected gesture as "correct" if (1) its class is correct and (2) it presents a time overlap of at least 50% with the ground truth gesture and vice versa (*50-50 Rule*).

Although recognition accuracy corresponds exactly to Recall (Eq. 6.2), recognition reliability differs slightly from Precision (Eq. 6.1), due to the appearance of false negatives in the denominator of Recall. Nevertheless, it is still possible to compute a modified *F1 score* as:

$$F1 = 2 \cdot \frac{Rec. \cdot Rel.}{Rec. + Rel.} \tag{7.14}$$

Please note that this definition does not correspond exactly to that of original F1 score used in gesture verification (Eq. 6.3). However, it shows similar properties, since it is equal to 1 (or 100%) for an optimal spotting system and 0 in the worst case.

7.3.2 Results using the complete model

As a first step, we evaluated our single gesture spotting approach using complete subgesture relationships (*Complete subgesture model*), as described in section 7.2.1.

Tables 7.4 – 7.5 summarize experimental settings and recognition results for all continuous datasets. We observe that our approach achieved satisfactory results on all datasets. Compared to previous works, we achieved 92.3%(277/300) accuracy on the EasyDigits dataset, versus 94.6%(284/300) of [2]. In this comparison, we should note that the best result in [2] has been achieved by retaining between 4 and 6 hand candidate locations in each frame, while our approach keeps only one such location. When only one location is kept, accuracy of [2] drops to 75.6%, which is equivalent to that achieved by Oka ([63]). We achieved 94.3%(66/70) recognition accuracy on the KinectStreamOrdered dataset, and 93.5%(362/387) on KinectStreamRepeat, i.e. 93.7%(428/457). In the original work [10], Frolova et al. achieved 93.9%(200/213). Due to the different subsets used, comparison can not be conclusive, although it indicates similar performance of the two approaches.

TABLE 7.2: Time delays on gesture spotting for the EasyDigits dataset

Digit	**Average Δt**	**Min Δt**	**Max Δt**
0	66	23	83
1	103	87	113
2	62	45	72
3	51	26	71
4	85	67	98
5	52	31	75
6	60	40	80
7	86	75	97
8	28	4	50
9	62	41	83

Tables 7.4 – 7.5 also show mean and standard deviation of the time delay, Δt, between ending time of spotted gesture and its announcement by the spotting module. Such time delay is inherent to indirect spotting methods, caused by the usage of a list of candidate gestures. We observe that Δt varies from dataset to dataset and that it is comparable to window lengths. In EasyDigits, $\Delta t = 65$ frames, corresponding to 2.2 seconds (capture is at a frame rate of 30 fps), which should be tolerable in most applications. Similarly, $\Delta t \approx$ 130 frames for the 6D datasets on average, corresponding to 2.5 seconds (sampling is at 60 fps). Table 7.2 shows analytical results for each gesture category for the EasyDigits dataset. We note that the longest delays appear in "subgesture" digits, i.e. digits that can be easily mistaken as parts of longer gestures, such as "1, 4, 7". This result is natural, since an expert system needs to wait to make sure that short gestures are not parts of a longer gesture.

Regarding substitutions, we didn't notice many systematic errors in our results. In original EasyDigits, "7" was substituted by "1" (3/30), "6" by "0" (2/30) and '9" by "1" (2/30). In synthetic EasyDigits (Case I, mixed), "9" corresponded to the worst case of correct spotting (2353/3000 or 78%), being substituted by "1" (204/3000), "5" (201/3000) and "7" (152/3000). In Upper6D (Case I, mixed), letter "I" was substituted by "J" in 7% of the cases (280/3900). The only major issue of concern appeared in Upper6D Case I, ordered), where letter "I" was substituted by "H" half of the times (1537/3000). Since the combination "HI" is quite common in many English words (e.g. "this", "think", "something"), this could be a serious problem of a practical spotting system.

Insertion errors affect severely recognition reliability (Eq. 7.13). To better understand the main sources of such errors, we repeated 4 of our spotting experiments on the synthetic datasets (Case I, mixed) for EasyDigits, Digits6D, Lower6D and Upper6D,

keeping the results of the intermediate sliding window searches, i.e. the gesture category found and the gesture verification label (*valid/invalid*). Due to the excessive number of results, we limited this experiment only to the first user of each dataset. Table 7.3 shows the frequency of appearance for the major surviving candidates in each dataset, as well as the corresponding insertion errors as percentages of the total number of insertions. As we observe, the large number of downward–motion gestures (such as digit "1", lower–case letter "ℓ" and upper–case letters "I" and "J") in the list of candidate gestures appears to be positively related to insertion errors.

TABLE 7.3: Major surviving candidates during gesture spotting

Dataset	Surviving candidates	Insertion errors
EasyDigits (Mixed)	1 (53%)	1 (67%), 4 (13%)
Digits6D (Mixed)	1 (43%)	1 (97%)
Lower6D (Mixed)	ℓ (22%), c (12%), i, j (14%)	ℓ (50%), c (28%), i, j (6%)
Upper6D (Mixed)	I,J (41%), C (13%)	I,J (61%), C (4%)

Regarding the order of appearance, we noticed that subgesture conditional probabilities collected from mixed sequences can provide high performance for ordered sequences, too. Nevertheless, evaluation on mixed sequences may offer greater insight about the recognition accuracy, reliability and time delay of a gesture recognition system. Although we didn't cover each random permutation explicitly, due to the exponential size of the problem, we considered a relatively large number of testing sequences. A peculiar phenomenon may be seen in Case I for upper–case letters, which shows a drop of 3% in recognition accuracy when compared to performance on mixed letters. Through further investigation, we found out that the drop is mainly due to two errors: deletion of letter "J" (1625 out of 3000 times) and substitution of letter "I" by "H" (1537/3000), as described above. Although the combination "IJ" is not very frequent in English language, it could be a serious problem in other languages. Please note again that these two problems appeared only in this ordered dataset and disappeared in its corresponding mixed version.

We notice that overwriting (Case I) outperforms "writing towards infinity" (Case II) by $3\% - 8\%$, with the exception of the artificially created continuous Digits6D streams (Ordered/Mixed). Through further investigation, we found out that low performance of Case II compared to Case I in EasyDigits is due to consistent substitutions of digit "1" by "7" and "4" when "1" is preceded by either "0" or "8". This is in fact expected if one considers the trajectory of a hand when drawing consecutively digits "0" and "1", as shown in Fig. 7.6–a, where it is virtually impossible to distinguish between a "genuine" 7, attached to "0", versus a "1", connected to "0" via a horizontal short line. Such a

(a) b)

FIGURE 7.6: (a) Confusing digit "1" to "7" after recognition of digit "0". (b) Avoiding confusion due to different "7" shape in Digits6D datastream [3].

phenomenon is not present when Case I (overwriting) is used, and interestingly enough, not present for the Digits6D sequence because of the different pattern used to draw the digit "7", as also shown in Fig. 7.6–b. One quick way to partially address this issue would be to impose a certain minimum time–distance between gestures for continuous gesture recognition, thus creating a "dead–time" between spotted gestures.

The only exception to the above rule is for the Digits6D datasets, where we notice that Reliability (Rel.) is worse in Case I, while Recognition accuracy (Rec.) remains almost the same. By inspecting the analytical spotting results, we saw that this issue is almost entirely due to double insertions of digit "1" in Case II. By measuring the frequencies of appearance for surviving gesture candidates (as in Table 7.3), we saw that digit "1" appears 60% more often in Digits6D – Case I than in Case II, thus explaining to a great extend this problem.

While our results are quite good for single gesture spotting applications, we would like to estimate a theoretical maximum that a *perfect* conflict resolution method can achieve, and thus understand the limits imposed by earlier components of our method for isolated gesture recognition and verification. To this end, we counted the cases where the "correct" candidate was found in the candidate list and reported it under *Possible Rec.* columns in Tables 7.4 – 7.5. We observe that in most cases there is room for improvement, indicating that advanced gesture verification and conflict resolution techniques affect significantly the spotting results.

Comparing *Possible Rec.* to isolated recognition accuracies (Tables 5.2 – 5.6), we notice that gesture spotting may actually perform better, assuming an ideal conflict resolution module. This is in contrast to the most common belief that isolating gestures always results into higher recognition accuracy. Intuitively, this result may be explained by the continuous nature of data in gesture spotting; indeed, isolated recognition assumes known –but also fixed– gesture time boundaries, while spotting allows choosing the most prominent ones. As a result, shifting gesture boundaries even by a few frames may lead to recognizing the correct gesture category, thus making the correct candidate available in conflict resolution rule during gesture spotting.

Overall, our results suggest that our approach is meaningful and can be applied in many scenarios and vocabulary sets. Spotting digits seems to be easier than spotting letters, while using position sensors instead of cameras results in higher performance, due to more accurate hand coordinates. We also saw that indirect gesture spotting techniques inherently exhibit some time delay in reporting the spotting results, as well as insertion errors. On the other hand, they may actually result in higher recognition accuracy, provided that a perfect conflict resolution method is available, outperforming isolated recognition and direct spotting methods in that ideal case. Finally, establishing clear protocols, such as imposing pauses between gestures or using gestures that avoid subgesture problems can improve overall system performance significantly, although they may compromise naturalness in user interaction.

7.3.3 Results using alternative conflict resolution methods

We also evaluated the three alternative methods for conflict resolution (Sec. 7.2.1.3), i.e. *class–only model*, *starting–time model* and *max PT score*.

Tables 7.6 – 7.7 summarize experimental settings and recognition results when the *class–only model* is used, i.e. when only interactions between gesture classes are modelled. Comparing to Tables 7.4 – 7.5, we see that $F1$–scores of *class–only model* are around 0.7% lower than those of the *complete subgesture model* for digits, with differences being smaller for the digits6D datasets.

Regarding lower and upper–case letters, using *class–only model* yields similar recognition accuracies (*Rec.*) to those of *complete subgesture model*, but at the cost of 1% drop in system's reliability (*Rel.*), leading to slightly lower $F1$–scores. The same model seems to *fix* the problem of ordered Upper6D letters (Case II), which now show recognition accuracy equal to that on random Upper6D letters (Case II).

Tables 7.8 – 7.9 summarize experimental settings and recognition results when the *starting–time model* is used, i.e. when interactions between gesture classes and starting times are modelled. By inspecting the results, we see that performances are around 1% lower than those achieved using the *complete subgesture model*. Compared to the *class–only model*, recognition accuracies for digits get improved by 1%, especially for the Computer Vision–based approaches (original and synthetic EasyDigits), while they remain almost unchanged for Digits6D, and slightly diminish for letters. This result suggests that probably more training gesture sequences are needed during subgesture model learning in the letter datasets.

Tables 7.10 – 7.11 summarize experimental settings and recognition results when the *max PT score* rule is used for conflict resolution, i.e. when we simply spot and report the gesture candidate showing the highest PT score. It is quite surprising that results for Case I are very close to those achieved by the other three methods. The only exception is for upper–case letters, where we notice around $6-10\%$ drop in $F1$–score. This method also proves insufficient for the Case II datasets, where we notice drops of around 5% for EasyDigits and lower6D, 10% for upper6D, but essentially no drop for digits6D. Thus, when Case I is strictly used, the *max PT score* rule could be used quite effectively and efficiently for spotting digits and lower–case letters, since it requires less memory compared to the other three methods, which would need to store the subgesture relationship probabilities.

Finally, it is worth noting that we also explored spotting the gesture candidate with the highest cosine similarity score, which however yielded very low recognition accuracies and reliabilities for all datasets (around 65% for digits and 27% for lower and upper–case letters), proving that a more sophisticated conflict resolution scheme is necessary for robust results.

7.3.4 Results using depth–only search fastNN

Additionally, we evaluated the performance of Depth–Only Search fast Nearest Neighbor (DOS–fastNN) [106], which offers increased computational efficiency compared to standard fastNN. As we showed in Sec. 5.3.3 – 5.3.4, DOS–fastNN performs worse than fastNN in isolated recognition, providing lower recognition accuracies in all datasets. Specifically, drop in accuracy may be low ($2-3\%$ in EasyDigits, digits6D and upper6D) or high (7% in lower6D, 12% in KinectT), suggesting that DOS–fastNN would provide moderate to bad spotting results. However, one should always keep in mind the continuous nature of data in gesture spotting; indeed, isolated recognition assumes known –but also fixed– gesture time boundaries, while spotting allows choosing the most prominent ones. As a result, moving gesture boundaries even by a few frames may lead to recognizing the correct gesture category, thus making the correct candidate available in conflict resolution rule during gesture spotting.

Tables 7.14 – 7.15 summarize experimental settings and recognition results when DOS–fastNN is used along with the *max PT score* rule for conflict resolution. Comparing with Tables 7.10 – 7.11, we observe around 1% drop in F1–score in most cases, with the exceptions of EasyDigits (2% drop in Case I), KinectStreamRepeat (5% drop) and Digits6D (almost zero drop in most cases). We observed similar drops when we used the *complete subgesture model* instead of the *max PT score* conflict resolution rule.

7.3.5 Effect of resampling parameter N

As described in Sec. 4.3.1, our gesture representation keeps only N trajectory points, through standard linear downsampling. Although we chose $N = 8$ for our experiments, one might question the performance of our system at higher N values. Indeed, as Fig. 5.2 suggests, choosing $N = 12$ may result into 3% better isolated recognition accuracy on the GreenDigits dataset, for relatively low noise levels ($SNR > 20$ dB). On the other hand, gesture verification when $N = 8$ performs slightly better than when $N > 8$, as we showed in Sec. 6.3.6.

To this end, we repeated our spotting experiments using $N = 12$ for the *max PT score* conflict resolution rule. However, in some datasets we noticed that a lot of gestures were falsely rejected by the spotter, mainly because they caused a higher number of fastNN backtrackings. Thus, we increased the value of threshold T_B by 50% on all datasets. Although this solution may not be the optimal one, it improved results in a universal way, avoiding parameter over–tuning.

Tables 7.12 – 7.13 summarize experimental settings and recognition results for this case. We observe that in some cases F1–score slightly increased or decreased compared to the case of $N = 8$. Specifically, we see improved results for Case II and decreased results for Case I, with the exception of the digits6D dataset. The most important improvement seems to be in the lower6D dataset (3%). However, performance could be improved for more appropriate threshold values.

To remove the effect of the T_B threshold, we also evaluated the performance of Depth–Only Search fast Nearest Neighbor (DOS–fastNN) [106] for $N = 12$. Our experimental results for this case are shown in Tables 7.16 – 7.17. Comparing to the results of $N = 8$ (Tables 7.14 – 7.15), we notice that choosing $N = 12$ improves results in all datasets, with the only exception of upper6D. Once again, the most important improvement seems to be in the lower6D dataset (3%).

Based on the above results, we cannot reach easy conclusions: choosing a higher N value may improve or worsen slightly the spotting results, depending on the nature of gesture data. The only promising result was that of lower–case letters, which presented significantly improved results for $N = 12$. Further investigation of the role of dimensionality is included in our plans for future work.

TABLE 7.4: Experimental results on continuous datasets, Case I – Writing on a 2D imaginary square (*complete subgesture model*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
GreenDigits	30	270	45	0.9	$[20, 30, \ldots, 110]$	130	99.7	98.0	95.1	96.6	62	25
EasyDigits	30	270	45	0.9	$[20, 30, \ldots, 110]$	130	98.0	92.3	82.4	87.1	65	23
KinectStreamOrdered	7	800	80	0.9	$[10, 20, \ldots, 110]$	130	100.0	94.3	93.0	93.6	85	13
KinectStreamRepeat	47	800	80	0.9	$[10, 20, \ldots, 110]$	130	98.0	93.5	80.1	86.3	80	15
EasyDigits (Ordered)	3000	270	45	0.9	$[10, 20, \ldots, 110]$	130	95.2	93.6	87.9	90.7	75	23
EasyDigits (Mixed)	3000	270	45	0.9	$[10, 20, \ldots, 110]$	130	96.8	93.6	91.3	92.4	75	23
Digits6D (Ordered)	1800	500	90	0.9	$[40, 60, \ldots, 220]$	260	99.2	97.3	90.0	93.5	142	35
Digits6D (Mixed)	1800	500	90	0.9	$[40, 60, \ldots, 220]$	260	99.3	97.3	93.4	95.3	144	40
Lower6D (Ordered)	1500	1040	150	0.9	$[30, 40, \ldots, 160]$	200	99.3	93.6	91.3	92.4	110	29
Lower6D (Mixed)	1500	1040	150	0.9	$[30, 40, \ldots, 160]$	200	98.9	93.5	90.9	92.2	111	28
Upper6D (Ordered)	3900	1248	200	0.9	$[20, 40, \ldots, 200]$	240	96.9	89.8	85.7	87.7	122	41
Upper6D (Mixed)	3900	1248	200	0.9	$[20, 40, \ldots, 200]$	240	98.1	92.7	88.2	90.4	126	42

TABLE 7.5: Experimental results on continuous datasets, Case II – Writing towards infinity (*complete subgesture model*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
EasyDigits (Ordered)	30	270	45	0.9	$[10, 20, \ldots, 110]$	130	94.8	85.5	79.7	82.5	66	24
EasyDigits (Mixed)	30	270	45	0.9	$[10, 20, \ldots, 110]$	130	94.9	87.0	83.2	85.1	66	26
Digits6D (Ordered)	7	800	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.3	96.2	97.3	141	33
Digits6D (Mixed)	3000	270	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.2	95.8	97.0	144	37
Lower6D (Ordered)	3000	270	150	0.9	$[30, 40, \ldots, 160]$	200	98.0	90.4	87.8	89.1	105	26
Lower6D (Mixed)	1800	500	150	0.9	$[30, 40, \ldots, 160]$	200	98.1	91.3	87.8	89.5	105	27
Upper6D (Ordered)	1800	500	200	0.9	$[20, 40, \ldots, 200]$	240	96.0	88.8	83.6	86.1	119	43
Upper6D (Mixed)	1500	1040	200	0.9	$[20, 40, \ldots, 200]$	240	96.4	89.0	84.0	86.4	119	43

TABLE 7.6: Experimental results on continuous datasets, Case I – Writing on a 2D imaginary square (*class–only model*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
GreenDigits	30	270	45	0.9	$[20, 30, \ldots, 110]$	130	99.7	98.3	95.5	96.9	62	25
EasyDigits	30	270	45	0.9	$[20, 30, \ldots, 110]$	130	98.0	92.3	82.7	87.2	65	23
KinectStreamOrdered	7	800	80	0.9	$[10, 20, \ldots, 110]$	130	100.0	94.3	93.0	93.6	85	13
KinectStreamRepeat	47	800	80	0.9	$[10, 20, \ldots, 110]$	130	98.0	94.8	79.6	86.5	80	15
EasyDigits (Ordered)	3000	270	45	0.9	$[10, 20, \ldots, 110]$	130	95.2	92.5	87.1	89.7	75	23
EasyDigits (Mixed)	3000	270	45	0.9	$[10, 20, \ldots, 110]$	130	96.8	92.6	90.2	91.4	74	23
Digits6D (Ordered)	1800	500	90	0.9	$[40, 60, \ldots, 220]$	260	99.2	96.7	89.4	92.9	143	35
Digits6D (Mixed)	1800	500	90	0.9	$[40, 60, \ldots, 220]$	260	99.3	97.5	93.5	95.5	144	40
Lower6D (Ordered)	1500	1040	150	0.9	$[30, 40, \ldots, 160]$	200	99.3	93.5	91.1	92.3	110	29
Lower6D (Mixed)	1500	1040	150	0.9	$[30, 40, \ldots, 160]$	200	98.9	93.1	90.2	91.6	111	28
Upper6D (Ordered)	3900	1248	200	0.9	$[20, 40, \ldots, 200]$	240	98.1	92.5	88.2	90.3	123	42
Upper6D (Mixed)	3900	1248	200	0.9	$[20, 40, \ldots, 200]$	240	98.2	92.5	87.8	90.1	126	42

TABLE 7.7: Experimental results on continuous datasets, Case II – Writing towards infinity (*class–only model*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
EasyDigits (Ordered)	30	270	45	0.9	$[10, 20, \ldots, 110]$	130	94.8	84.4	78.3	81.2	65	25
EasyDigits (Mixed)	30	270	45	0.9	$[10, 20, \ldots, 110]$	130	94.9	85.8	81.5	83.6	66	26
Digits6D (Ordered)	7	800	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.1	96.0	97.0	141	33
Digits6D (Mixed)	3000	270	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.0	95.7	96.9	144	37
Lower6D (Ordered)	3000	270	150	0.9	$[30, 40, \ldots, 160]$	200	98.1	90.3	87.6	88.9	105	26
Lower6D (Mixed)	1800	500	150	0.9	$[30, 40, \ldots, 160]$	200	98.0	91.2	87.6	89.4	105	27
Upper6D (Ordered)	1800	500	200	0.9	$[20, 40, \ldots, 200]$	240	96.1	88.1	82.6	85.3	119	43
Upper6D (Mixed)	1500	1040	200	0.9	$[20, 40, \ldots, 200]$	240	96.4	88.0	82.9	85.3	118	43

TABLE 7.8: Experimental results on continuous datasets, Case I – Writing on a 2D imaginary square (*starting–time model*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
GreenDigits	30	270	45	0.9	$[20, 30, \ldots, 110]$	130	99.7	97.7	94.5	96.1	62	25
EasyDigits	30	270	45	0.9	$[20, 30, \ldots, 110]$	130	98.0	93.3	83.3	88.1	65	23
KinectStreamOrdered	7	800	80	0.9	$[10, 20, \ldots, 110]$	130	100.0	91.4	90.1	90.8	85	13
KinectStreamRepeat	47	800	80	0.9	$[10, 20, \ldots, 110]$	130	98.0	94.3	79.7	86.4	80	15
EasyDigits (Ordered)	3000	270	45	0.9	$[10, 20, \ldots, 110]$	130	95.2	93.1	87.3	90.1	75	23
EasyDigits (Mixed)	3000	270	45	0.9	$[10, 20, \ldots, 110]$	130	96.8	92.9	90.3	91.6	75	23
Digits6D (Ordered)	1800	500	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	97.1	89.6	93.2	143	35
Digits6D (Mixed)	1800	500	90	0.9	$[40, 60, \ldots, 220]$	260	99.3	97.3	93.2	95.2	144	40
Lower6D (Ordered)	1500	1040	150	0.9	$[30, 40, \ldots, 160]$	200	99.3	92.5	90.2	91.3	111	29
Lower6D (Mixed)	1500	1040	150	0.9	$[30, 40, \ldots, 160]$	200	98.9	92.7	89.9	91.2	112	28
Upper6D (Ordered)	3900	1248	200	0.9	$[20, 40, \ldots, 200]$	240	98.1	91.4	87.1	89.2	123	42
Upper6D (Mixed)	3900	1248	200	0.9	$[20, 40, \ldots, 200]$	240	98.2	92.4	87.7	90.0	126	42

TABLE 7.9: Experimental results on continuous datasets, Case II – Writing towards infinity (*starting–time model*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
EasyDigits (Ordered)	30	270	45	0.9	$[10, 20, \ldots, 110]$	130	94.8	84.2	78.2	81.1	65	24
EasyDigits (Mixed)	30	270	45	0.9	$[10, 20, \ldots, 110]$	130	94.9	85.7	81.6	83.6	66	26
Digits6D (Ordered)	7	800	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.1	96.1	97.1	141	33
Digits6D (Mixed)	3000	270	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.1	95.6	96.8	144	36
Lower6D (Ordered)	3000	270	150	0.9	$[30, 40, \ldots, 160]$	200	98.0	90.2	87.2	88.7	105	26
Lower6D (Mixed)	1800	500	150	0.9	$[30, 40, \ldots, 160]$	200	98.0	90.9	87.2	89.0	105	27
Upper6D (Ordered)	1800	500	200	0.9	$[20, 40, \ldots, 200]$	240	96.1	88.3	83.0	85.6	119	43
Upper6D (Mixed)	1500	1040	200	0.9	$[20, 40, \ldots, 200]$	240	96.6	88.5	83.4	85.9	119	43

TABLE 7.10: Experimental results on continuous datasets, Case I – Writing on a 2D imaginary square (*max PT score*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
GreenDigits	30	270	45	0.9	$[20, 30, \ldots, 110]$	130	99.7	98.0	95.1	96.6	62	25
EasyDigits	30	270	45	0.9	$[20, 30, \ldots, 110]$	130	98.0	92.3	82.7	87.2	65	23
KinectStreamOrdered	7	800	80	0.9	$[10, 20, \ldots, 110]$	130	100.0	94.3	91.7	93.0	89	13
KinectStreamRepeat	47	800	80	0.9	$[10, 20, \ldots, 110]$	130	99.5	94.3	80.3	86.7	80	15
EasyDigits (Ordered)	3000	270	45	0.9	$[10, 20, \ldots, 110]$	130	95.3	91.8	86.5	89.1	75	23
EasyDigits (Mixed)	3000	270	45	0.9	$[10, 20, \ldots, 110]$	130	96.8	91.6	89.1	90.3	74	23
Digits6D (Ordered)	1800	500	90	0.9	$[40, 60, \ldots, 220]$	260	99.2	97.1	89.8	93.3	142	35
Digits6D (Mixed)	1800	500	90	0.9	$[40, 60, \ldots, 220]$	260	99.3	97.6	93.7	95.6	144	40
Lower6D (Ordered)	1500	1040	150	0.9	$[30, 40, \ldots, 160]$	200	99.3	91.2	88.2	89.7	110	29
Lower6D (Mixed)	1500	1040	150	0.9	$[30, 40, \ldots, 160]$	200	98.9	91.0	87.8	89.4	111	28
Upper6D (Ordered)	3900	1248	200	0.9	$[20, 40, \ldots, 200]$	240	98.2	85.1	78.7	81.8	126	41
Upper6D (Mixed)	3900	1248	200	0.9	$[20, 40, \ldots, 200]$	240	98.2	84.3	77.5	80.7	128	42

TABLE 7.11: Experimental results on continuous datasets, Case II – Writing towards infinity (*max PT score*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
EasyDigits (Ordered)	30	270	45	0.9	$[10, 20, \ldots, 110]$	130	94.8	82.3	75.8	78.9	66	24
EasyDigits (Mixed)	30	270	45	0.9	$[10, 20, \ldots, 110]$	130	94.9	81.0	76.5	78.7	67	26
Digits6D (Ordered)	7	800	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.1	96.1	97.1	141	33
Digits6D (Mixed)	3000	270	90	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.3	96.0	97.1	144	36
Lower6D (Ordered)	3000	270	150	0.9	$[30, 40, \ldots, 160]$	200	97.8	85.7	82.6	84.2	106	27
Lower6D (Mixed)	1800	500	150	0.9	$[30, 40, \ldots, 160]$	200	97.5	86.2	82.3	84.2	105	27
Upper6D (Ordered)	1800	500	200	0.9	$[20, 40, \ldots, 200]$	240	95.1	79.1	72.0	75.4	121	42
Upper6D (Mixed)	1500	1040	200	0.9	$[20, 40, \ldots, 200]$	240	95.4	78.1	71.0	74.4	120	42

TABLE 7.12: Experimental results on continuous datasets, Case I – Writing on a 2D imaginary square (*max PT score* – $N = 12$)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
GreenDigits	30	270	70	0.9	$[20, 30, \ldots, 110]$	130	99.7	97.0	90.9	93.9	61	25
EasyDigits	30	270	70	0.9	$[20, 30, \ldots, 110]$	130	97.7	93.7	83.6	88.4	64	23
KinectStreamOrdered	7	800	120	0.9	$[10, 20, \ldots, 110]$	130	100.0	94.3	91.7	93.0	87	12
KinectStreamRepeat	47	800	120	0.9	$[10, 20, \ldots, 110]$	130	99.5	95.9	74.5	83.8	79	15
EasyDigits (Ordered)	3000	270	70	0.9	$[10, 20, \ldots, 110]$	130	96.0	90.8	84.1	87.3	74	23
EasyDigits (Mixed)	3000	270	70	0.9	$[10, 20, \ldots, 110]$	130	97.7	91.4	88.7	90.0	74	24
Digits6D (Ordered)	1800	500	140	0.9	$[40, 60, \ldots, 220]$	260	99.4	97.9	90.5	94.1	144	35
Digits6D (Mixed)	1800	500	140	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.1	94.2	96.1	146	39
Lower6D (Ordered)	1500	1040	230	0.9	$[30, 40, \ldots, 160]$	200	99.4	91.5	88.6	90.0	109	29
Lower6D (Mixed)	1500	1040	230	0.9	$[30, 40, \ldots, 160]$	200	99.4	91.4	88.2	89.8	109	27
Upper6D (Ordered)	3900	1248	300	0.9	$[20, 40, \ldots, 200]$	240	96.3	84.6	77.2	80.7	133	39
Upper6D (Mixed)	3900	1248	300	0.9	$[20, 40, \ldots, 200]$	240	96.4	84.2	76.8	80.3	134	38

TABLE 7.13: Experimental results on continuous datasets, Case II – Writing towards infinity (*max PT score* – $N = 12$)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
EasyDigits (Ordered)	30	270	70	0.9	$[10, 20, \ldots, 110]$	130	95.0	82.4	75.8	79.0	65	25
EasyDigits (Mixed)	30	270	70	0.9	$[10, 20, \ldots, 110]$	130	95.5	81.6	77.1	79.3	67	26
Digits6D (Ordered)	7	800	140	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.0	96.0	97.0	140	34
Digits6D (Mixed)	3000	270	140	0.9	$[40, 60, \ldots, 220]$	260	99.4	97.9	95.9	96.9	143	37
Lower6D (Ordered)	3000	270	230	0.9	$[30, 40, \ldots, 160]$	200	99.0	89.1	85.4	87.2	102	26
Lower6D (Mixed)	1800	500	230	0.9	$[30, 40, \ldots, 160]$	200	98.3	88.8	84.8	86.7	101	27
Upper6D (Ordered)	1800	500	300	0.9	$[20, 40, \ldots, 200]$	240	94.8	80.5	73.1	76.6	124	40
Upper6D (Mixed)	1500	1040	300	0.9	$[20, 40, \ldots, 200]$	240	94.9	80.1	72.5	76.1	124	40

TABLE 7.14: Experimental results on continuous datasets, Case I – Writing on a 2D imaginary square (DOS – *max PT score*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
GreenDigits	30	270	–	0.9	$[20, 30, \ldots, 110]$	130	99.7	97.3	91.5	94.3	60	25
EasyDigits	30	270	–	0.9	$[20, 30, \ldots, 110]$	130	98.3	92.0	82.4	86.9	63	23
KinectStreamOrdered	7	800	–	0.9	$[10, 20, \ldots, 110]$	130	100.0	94.3	83.5	88.6	87	13
KinectStreamRepeat	47	800	–	0.9	$[10, 20, \ldots, 110]$	130	99.2	94.3	80.9	87.1	79	15
EasyDigits (Ordered)	3000	270	–	0.9	$[10, 20, \ldots, 110]$	130	95.1	90.9	83.0	86.8	73	22
EasyDigits (Mixed)	3000	270	–	0.9	$[10, 20, \ldots, 110]$	130	96.8	91.0	87.2	89.1	73	24
Digits6D (Ordered)	1800	500	–	0.9	$[40, 60, \ldots, 220]$	260	99.4	97.5	90.2	93.7	143	35
Digits6D (Mixed)	1800	500	–	0.9	$[40, 60, \ldots, 220]$	260	99.3	97.5	93.6	95.5	144	40
Lower6D (Ordered)	1500	1040	–	0.9	$[30, 40, \ldots, 160]$	200	99.1	90.4	87.4	88.8	107	30
Lower6D (Mixed)	1500	1040	–	0.9	$[30, 40, \ldots, 160]$	200	98.7	90.2	86.9	88.5	107	29
Upper6D (Ordered)	3900	1248	–	0.9	$[20, 40, \ldots, 200]$	240	98.7	84.5	77.3	80.7	125	42
Upper6D (Mixed)	3900	1248	–	0.9	$[20, 40, \ldots, 200]$	240	98.6	83.6	76.0	79.6	128	42

TABLE 7.15: Experimental results on continuous datasets, Case II – Writing towards infinity (DOS – *max PT score*)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
EasyDigits (Ordered)	30	270	–	0.9	$[10, 20, \ldots, 110]$	130	94.5	81.7	75.2	78.3	66	25
EasyDigits (Mixed)	30	270	–	0.9	$[10, 20, \ldots, 110]$	130	95.3	80.6	75.7	78.1	67	26
Digits6D (Ordered)	7	800	–	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.0	96.0	97.0	141	34
Digits6D (Mixed)	3000	270	–	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.0	95.7	96.8	143	37
Lower6D (Ordered)	3000	270	–	0.9	$[30, 40, \ldots, 160]$	200	97.7	84.6	81.1	82.8	99	27
Lower6D (Mixed)	1800	500	–	0.9	$[30, 40, \ldots, 160]$	200	97.2	85.2	81.1	83.1	98	27
Upper6D (Ordered)	1800	500	–	0.9	$[20, 40, \ldots, 200]$	240	95.3	78.6	71.1	74.7	120	42
Upper6D (Mixed)	1500	1040	–	0.9	$[20, 40, \ldots, 200]$	240	95.4	77.3	69.9	73.4	118	42

TABLE 7.16: Experimental results on continuous datasets, Case I – Writing on a 2D imaginary square (DOS – *max PT score* – $N = 12$)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
GreenDigits	30	270	–	0.9	$[20, 30, \ldots, 110]$	130	99.3	97.0	90.1	93.4	61	25
EasyDigits	30	270	–	0.9	$[20, 30, \ldots, 110]$	130	98.0	93.7	83.4	88.2	64	22
KinectStreamOrdered	7	800	–	0.9	$[10, 20, \ldots, 110]$	130	100.0	94.3	83.5	88.6	87	13
KinectStreamRepeat	47	800	–	0.9	$[10, 20, \ldots, 110]$	130	99.7	96.1	76.4	85.1	79	15
EasyDigits (Ordered)	3000	270	–	0.9	$[10, 20, \ldots, 110]$	130	95.9	91.2	84.1	87.5	74	22
EasyDigits (Mixed)	3000	270	–	0.9	$[10, 20, \ldots, 110]$	130	97.3	91.0	87.9	89.4	74	23
Digits6D (Ordered)	1800	500	–	0.9	$[40, 60, \ldots, 220]$	260	99.4	97.8	90.6	94.0	144	35
Digits6D (Mixed)	1800	500	–	0.9	$[40, 60, \ldots, 220]$	260	99.4	97.8	94.0	95.9	146	39
Lower6D (Ordered)	1500	1040	–	0.9	$[30, 40, \ldots, 160]$	200	99.4	90.6	87.4	89.0	109	29
Lower6D (Mixed)	1500	1040	–	0.9	$[30, 40, \ldots, 160]$	200	99.4	90.3	87.0	88.6	109	27
Upper6D (Ordered)	3900	1248	–	0.9	$[20, 40, \ldots, 200]$	240	98.8	84.8	75.8	80.0	130	40
Upper6D (Mixed)	3900	1248	–	0.9	$[20, 40, \ldots, 200]$	240	98.7	84.4	75.2	79.6	132	40

TABLE 7.17: Experimental results on continuous datasets, Case II – Writing towards infinity (DOS – *max PT score* – $N = 12$)

Dataset	**Test sequences**	**Tree examples**	T_B	T_P	**Windows**	L	***Possible Rec.*** *(%)*	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)	$\mu_{\Delta t}$	$\sigma_{\Delta t}$
EasyDigits (Ordered)	30	270	–	0.9	$[10, 20, \ldots, 110]$	130	94.6	81.9	75.5	78.6	66	25
EasyDigits (Mixed)	30	270	–	0.9	$[10, 20, \ldots, 110]$	130	95.0	81.2	76.6	78.8	67	26
Digits6D (Ordered)	7	800	–	0.9	$[40, 60, \ldots, 220]$	260	99.4	98.1	96.1	97.1	140	34
Digits6D (Mixed)	3000	270	–	0.9	$[40, 60, \ldots, 220]$	260	99.4	97.8	95.7	96.7	143	37
Lower6D (Ordered)	3000	270	–	0.9	$[30, 40, \ldots, 160]$	200	99.0	88.2	85.1	86.6	100	27
Lower6D (Mixed)	1800	500	–	0.9	$[30, 40, \ldots, 160]$	200	98.5	87.9	84.2	86.0	100	27
Upper6D (Ordered)	1800	500	–	0.9	$[20, 40, \ldots, 200]$	240	95.7	79.1	69.2	73.8	118	41
Upper6D (Mixed)	1500	1040	–	0.9	$[20, 40, \ldots, 200]$	240	95.5	78.2	68.2	72.9	117	41

7.3.6 Character spotting on real texts

While evaluating spotting algorithms on ordered and random sequences makes sense for digit recognition purposes, letters usually appear in forms of words, obeying specific grammatical rules. This property makes several letter pairs more frequent than others, and it could possibly affect spotting performance.

To test this hypothesis, we segmented a real English text into 900 small sentences of 5 words each –we useed upper–case letters– and ran all spotting algorithms again [5]. Additionally, we repeated the above process for a real Italian text [6]. Connections between consecutive letters were created artificially, based on the *overwriting* mode (*Case I – Writing on a 2D imaginary square*).

Our results, depicted in Tables 7.18–7.19, suggest a 2% recognition accuracy (*Rec.*) drop between random and real sequences of capital letters. More severely, we observe a 7% drop in system's reliability (*Rel.*). However, we should keep in mind that subgesture relationships were learned by random sequences and not by texts of the same language. Further exploration of this issue is included in our plans for future work.

TABLE 7.18: Recognition accuracies on typical English text

Method	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)
max PT	81.4	68.0	74.1
class–only	89.5	80.0	84.5
starting–time	89.9	80.7	85.1
complete subgesture	**90.6**	**81.2**	**85.6**

TABLE 7.19: Recognition accuracies on typical Italian text

Method	***Rec.*** (%)	***Rel.*** (%)	***F1*** (%)
max PT	80.9	66.8	73.2
class–only	88.7	78.4	83.2
starting–time	88.8	78.7	83.5
complete subgesture	**89.8**	**79.5**	**84.3**

7.4 Experimental results for word gesture spotting

In this section, we present our experimental results for gesture spotting of words on continuous hand coordinate streams.

[5] We used the first 4500 words of the short stories collection "The adventures of Sherlock Holmes", written by Sir Arthur Conan Doyle, 1892. The entire collection is in the public domain and is available through Project Gutenberg http://www.gutenberg.org/ebooks/1661.

[6] We used the first 4500 words of the epic poem "La Divina Commedia: Paradiso", written by Dante Alighieri, 1308–1321. The entire poem is on public domain and is available through Project Gutenberg http://www.gutenberg.org/ebooks/1011.

7.4.1 Datasets and experimental setup

To evaluate our approach for recognition of word gestures, we use the "Motion Words" dataset from the 6DMG database (Table 3.1), which contains continuous streams of 40 words, written on the air by 12 users (40 words, 5 examples/user/word, 2400 sequences in total). These sequences are split in 4 sets, each containing 10 words with an average of 4 characters per word, corresponding to 4 different scenarios of using gestures to control an integrated entertainment system, supporting selecting TV channels, music genres and basic internet websites, as shown in Table 3.2.

Besides the original sequences, we also conducted experiments on 4800 artificially created sequences (12 users, 40 words, 10 examples/user/word), using exactly the same dataset structure (4 sets, 10 words per set, following Table 3.2).

In total, we evaluated the following methods:

- *Isolated recognition*, which treats the trajectory of a word as a whole and applies standard isolate recognition using Maximum Cosine Similarity, as described in Chapter 5. Recognition accuracy of this approach depends on the number of trajectory points, N, kept after resampling.
- *Spotting + Levenshtein distance*, which first applies single gesture spotting (Sec. 7.2.1) and then uses the Levenshtein distance to recognize words (Sec. 7.2.2.1). We tested all four conflict resolution variations (Sec. 7.2.1.3), i.e. *max PT score*, *class–only model*, *starting–time model* and *complete subgesture model*.
- *Spotting + Weighted Edit distance*, which first applies single gesture spotting and then uses the Weighted Edit distance to recognize words (Sec. 7.2.2.1). Once again, we tested all four conflict resolution variations, as in the previous method.

7.4.2 Recognition accuracies of the various methods

Table 7.20 shows word recognition accuracies for the original *Motion words* dataset. We observe that all spotting methods perform quite well, with *starting–time model* providing the highest recognition accuracies on average. In all cases, Weighted Edit distance outperforms simple Levenshtein distance, as expected. In short, Edit distance variations provide quite high word recognition accuracies. As a comparison, the character recognition accuracies were about 78% on average. Clearly, operating at the word level allows for error correction, at the cost of additional delay in the announcement of spotting results.

It is quite surprising that standard isolated recognition provides the best results, even when only 32 trajectory points are kept after downsampling. However, we suspect that this result is highly dependent on the specific dataset, and would not generalize well if the user imposed different types of connecting strokes between single character gestures. Indeed, accuracy drops to 93% for the synthetic dataset(Table 7.21), as expected. On the other hand, spotting approaches present very stable behaviour, proving that they should be preferred over isolated recognition.

TABLE 7.20: Word recognition accuracies on the original *Motion Words* dataset

Method	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Avg.**
Isolated recognition $N = 16$	98.2	99.2	99.0	96.7	98.2
Isolated recognition $N = 32$	98.8	99.7	99.8	99.7	**99.2**
max PT + Levenshtein dist.	89.0	90.7	85.0	85.2	87.5
max PT + Weighted Edit dist.	95.2	99.0	95.5	97.3	96.8
class–only + Levenshtein dist.	94.7	95.2	94.3	93.3	94.4
class–only + Weighted Edit dist.	96.0	98.8	97.2	96.5	97.1
starting–time + Levenshtein dist.	95.7	96.8	95.7	94.8	**95.8**
starting–time + Weighted Edit dist.	98.2	99.3	98.0	98.0	**98.4**
complete subgesture + Levenshtein dist.	94.3	95.7	94.2	93.3	94.4
complete subgesture + Weighted Edit dist.	97.5	98.8	96.7	97.0	97.5

TABLE 7.21: Word recognition accuracies on the synthetic *Motion Words* dataset

Method	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Avg.**
Isolated recognition $N = 16$	91.4	93.5	89.8	87.7	90.6
Isolated recognition $N = 32$	90.9	95.3	93.5	92.2	**93.0**
max PT + Levenshtein dist.	93.0	95.8	90.0	90.3	92.3
max PT + Weighted Edit dist.	98.2	100.0	98.3	99.8	99.1
class–only + Levenshtein dist.	96.2	97.2	96.8	96.8	96.8
class–only + Weighted Edit dist.	99.3	99.7	99.3	99.8	99.5
starting–time + Levenshtein dist.	97.0	97.2	97.3	98.0	**97.4**
starting–time + Weighted Edit dist.	99.7	100.0	99.3	100.0	**99.8**
complete subgesture + Levenshtein dist.	95.5	95.5	95.8	91.8	94.7
complete subgesture + Weighted Edit dist.	99.7	100.0	99.2	99.8	99.7

7.4.3 Recognition accuracies for larger dictionaries

While Edit distance proved to be a very powerful method for word gesture recognition, we should keep in mind that the dictionary size was quite small (10 words), allowing for effective error correction and character disambiguation. Naturally, one could expect that performance would drop for larger dictionaries. To take an estimate of such drop, we repeated our experiments considering a dictionary of 40 words, through merging all

4 sets together. In Table 7.22, we show only the average word recognition accuracies, for brevity. As expected, accuracies drop, around 2% for the original dataset and 1% for the synthetic dataset. We find this result quite encouraging, considering that the dictionary size is 4 times bigger.

Regarding performances of the various spotting methods, results of Table 7.22 seem to agree with those of Tables 7.20 – 7.21, with richer models (*starting–time model*, *complete subgesture model*) and Weighted Edit distance providing the best results.

Regarding word confusions (in the original datasets), the most severe errors were caused by words which are quite similar: "ABC"–"BBC" (4/60), "FB"–'TBS" (6/60), "MLB"–"MAP" (3/60) and "TV"–"TNT" (5/60). Clearly, such mistakes make sense, and may be expected by most word recognition systems.

TABLE 7.22: Average word recognition accuracies on the *Motion Words* dataset with all 40 words in the dictionary

Method	**Original dataset**	**Synthetic dataset**
Isolated recognition $N = 16$	97.5	82
Isolated recognition $N = 32$	**98.5**	87.7
max PT + Levenshtein dist.	76.5	82.9
max PT + Weighted Edit dist.	93.8	97.0
class–only + Levenshtein dist.	87.3	92.9
class–only + Weighted Edit dist.	95.0	**98.8**
starting–time + Levenshtein dist.	89.3	93.7
starting–time + Weighted Edit dist.	**96.8**	**99.1**
complete subgesture + Levenshtein dist.	87.3	93.8
complete subgesture + Weighted Edit dist.	95.0	**99.2**

7.5 Discussion

In this chapter, we explored gesture spotting on continuous streams of hand coordinates, targeting detection and recognition of trajectories of digits and letters in an *indirect* way, i.e. without using motion cues or other marks to detect the starting and ending time boundaries of the performed gestures. Our approach is based on applying isolated classification through fastNN Maximum Cosine Similarity (fastNN–MCS), collecting groups of overlapping gesture candidates and applying conflict resolution methods to reach the final spotting result.

Our main contribution lies in proposing a novel method to perform conflict resolution, combining both the cosine similarity score and the time duration of the candidate gestures. Moreover, we introduced a probabilistic framework to learn and handle subgesture

relationships, extending previous models which used strict rules for the gesture classes [2]. Specifically, our approach models both categories and relative time boundaries of the gesture candidates. Finally, we also experimented with gesture spotting on real texts and used a properly modified version of the Weighted Edit Distance [123] for word recognition, which takes into account the confusions made during single gesture spotting, supporting small vocabularies of 10-40 words.

Chapter 8

Computational efficiency

Contents

Computational efficiency is an important issue in any gesture recognition system. All interactive applications require real–time gesture spotting, which may become quite challenging depending on the computing device (standard personal computer, mobile phone or other device). In certain cases, offline or delayed recognition can be afforded, typically during the training phase of a system.

The main goal of this work is to achieve high recognition accuracy and reliability at low computational cost. Clearly, there is a trade–off between these two targets, which

leads to a comparison between the state–of–the–art method, Dynamic Time Warping (DTW), and and the method we chose, Maximum Cosine Similarity (MCS). As we will show in this chapter, MCS offers a significant computational speedup, while at the same time the loss in accuracy is minor (Chapter 5). This speedup is largely boosted by tree-based fast Nearest Neighbor (fastNN) [106] and Single–Instruction–Multiple-Data (SIMD) techniques [126].

In short, the major contributions of this chapter are:

- Providing a fair computational efficiency comparison between DTW and MCS on two isolated gesture datasets, exploring various parameters, such as effect of fastNN and SIMD instructions.
- Proposing a novel method to speedup DTW, using fastNN for initialization.
- Exploring performance of fastNN with varying number of training examples and gesture classes.
- Building a real–time gesture spotting application.

In all cases, we conduct thorough experiments on real and synthetic gesture datasets.

8.1 Computational complexity of various recognition methods

8.1.1 Using fast Nearest Neighbor in gesture spotting

As described in chapters 5 – 7, our approach relies heavily on the tree-based fast Nearest Neighbor (fastNN) algorithm of Katsavounidis et al.[106], which offers an exact solution in nearly logarithmic time. This gain significantly reduces the computational time needed for classification of new gesture candidates during the spotting procedure.

Nevertheless, our gesture spotting approach uses a sliding window over a continuous stream of hand coordinates, resulting in a large number of fastNN searches in order to form groups of gesture candidates. Since sparsity is an inherent feature of gesture spotting, most of these candidates are invalid (out–of–vocabulary or almost random trajectories) and they typically compromise the performance of fastNN (Chapter 6), causing a large number of tree backtrackings.

To address this issue, we first set a threshold to limit the number of backtrackings up to a certain level, rejecting as invalid all candidates which exceed that threshold (*Rule2* in

algorithm 1, Chapter 5). Although this method is inherently *lossy*, in terms of rejecting some valid inputs as well, our spotting results suggest that this loss is quite low (about 2% – see column *Possible Rec. (%)* in Tables 7.4 – 7.11). As we will show later in this chapter (Sec. 8.3.3), this technique also results into significant computational gain (4× speedup).

8.1.2 Improving DTW using fastNN

One way to improve the computational efficiency of DTW search is by using a good initial estimate of the global DTW distance. Ideally, the best such estimate would be the true global minimum DTW distance, since then most of the vectors would be eliminated through the *LBKeogh* lower bound [107].

In this work, we first perform a fastNN search and locate the vector showing the minimum Euclidean Distance. We then compute the corresponding DTW distance and use it as an initial estimate. A nice property of fastNN is that its initialization phase depends only on the training examples and not on the query vector at all (as compared to *LBKeogh*). Thus, its cost is fixed, since it is performed only once, before the search starts and is thus negligible when a large amount of query vectors is tested.

Our experimental results (Sec. 8.3.2) suggest that the total cost of the fastNN search and *LBKeogh*–DTW can provide a speedup of $1.3 - 1.5\times$, depending on the dataset. To our knowledge, such initialization of DTW is novel. Although using the gesture of minimum Euclidean Distance as the initial candidate may sound trivial, a straight–forward application of full search is not computationally efficient. In contrast, fastNN allows for a more efficient implementation, decreasing the initialization computational cost at a significant level.

8.1.3 Support Vector Machines

For our experiments involving Support Vector Machines (SVM), we used *LIBSVM* [119], a standard publicly available library, which offers efficient and robust implementations of various SVM–related tasks, including training and testing. We improved computational efficiency further, by precomputing the hyperplanes for the case of linear kernel (Eq. 5.7 – 5.8) and applying some standard programming optimizations in the probability estimation part. In total, our modified version of *LIBSVM* shows a speedup of 5.8×.

8.2 Comparing isolated recognition performances

8.2.1 Methods and Datasets

In order to provide a fair comparison between our approach and other recognition methods, we ran thorough experiments on the three Nintendo Wii 6D datasets, containing digits (Digits6D) lower–case (Lower6D) and upper–case letters (Upper6D).

To make comparison fair, we considered only isolated recognition, since we did not have any optimized implementation of DTW for spotting [2]. Thus, we evaluated the performance of Maximum Cosine Similarity (MCS), which forms the basis of our gesture recognition system. We also experimented with Depth-Only-Search MCS (DOS-MCS), fastKNN–MCS Support Vector Machines (SVM), standard Dynamic Time Warping (DTW) and its low–complexity variant DTW8.

8.2.2 Experimental setup

Our experiments were run on a typical desktop environment, consisting of an Intel(R) Core^{TM} i7 CPU running Microsoft (R) Windows^{TM} 7 64–bit OS. We implemented all methods in C, applying some basic code optimization and compiled them using Microsoft (R) Visual Studio 2010, Release configuration, *-Od* compiler–optimized executable, to make results as platform–independent as possible. Wherever applicable, we optimized our code by using Single–Instruction–Multiple-Data (SIMD) techniques, namely using the "SSE2" enhanced instruction set, which is supported by most modern Intel CPU IA-32 and Intel-64 architectures [126].

In order to better differentiate and understand the factors that can contribute to higher computation efficiency, we created the following variants of the 2 basic algorithms (DTW8 and MCS):

- DTW8: unconstrained DTW with $N = 8$
- DTW8–Sakoe: DTW8 with *Sakoe-Chiba band* [113]
- DTW8–LBKeogh: DTW8-Sakoe with *LBKeogh* lower bound [107]
- DTW8–fastNN: DTW8–LBKeogh, initialized using fastNN (Sec. 8.3.2)
- model–DTW: probabilistic variant of DTW [2], as described in Sec. 5.2.1.1
- MCS: maximum cosine similarity (MCS) with $N = 8$

- MCS–Partial: MCS with partial distance calculation [127]
- MCS–fastNN: MCS using fastNN as per [106]
- MCS–DOS: depth-only-search using fastNN as per [106]
- MCS–fastKNN5: extended version fastNN, which finds the 5 nearest neighbors (Sec. 5.1.3)

Additionally, we evaluated the performance of Support Vector Machines (SVM), using our improved version of the *LIBSVM* library [119] (Sec. 8.1.3).

8.2.3 Experimental results

Tables 8.1 and 8.2 show speedups for 6 methods, under both standard–C and SSE2 implementations, for the digits6D and lower6D datasets respectively. For clarity, we normalized results using full-search MCS as point of reference.

Overall, MCS executes much faster than DTW, as expected by theoretical computational complexity ($O(N)$ versus $O(N^2)$), while it is easily vectorized using Intel's SSE2 intrinsics [126]. Applying further constraints, such as *Sakoe-Chiba band* [113] and *LB Keogh* [107], improved DTW performance by about 3×, but even the fastest DTW implementation executed 5 times slower than the slowest (full–search) MCS.

On the other hand, *model–DTW*, the probabilistic variant of DTW [2], shows a fixed computational cost, which is proportional to the number of gesture categories, while it is independent of the number of training examples. Thus, it executed $2-6$ times faster than full–search MCS, although around 7 times slower than MCS–fastNN and up to 40 times more slowly than MCS–DOS. However, model–DTW presented decreased recognition accuracies in our isolated recognition experiments (Sec. 5.3.3 – 5.3.4).

Regarding MCS, we observe that fastNN greatly speeds up the search process, especially for the larger Upper6D dataset, as expected [106, 121]. Specifically, the fastNN tree contains 500 training examples for digits6D, and 3120 examples for upper6D. In the next section, we explore this behaviour in detail.

SSE2 further improves results, although its impact is rather small – around 2× instead of the theoretical maximum 4×, due to the limited number of arithmetic versus logical operations involved in tree searching. An additional feature of MCS is that it allows applications to trade–off an additional $3-10\times$ acceleration for a modest $(1-10\%)$ drop in recognition accuracy through the depth-only-search version (MCS–DOS).

TABLE 8.1: Execution time speedup for all methods (digits6D dataset)

Method	Standard C	SSE2
DTW8	$\frac{1}{18}\times$	–
DTW8–Sakoe	$\frac{1}{13}\times$	–
DTW8–LB_{Keogh}	$\frac{1}{6}\times$	–
model–DTW	2.3×	–
MCS	1×	2.1×
MCS–Partial	2.1×	2.5×
MCS–fastNN	17.2×	28.3×
MCS–DOS	48.2×	77.3×
MCS–fastKNN5	8.1×	8.3×
SVM	6.5×	6.5×

TABLE 8.2: Execution time speedup for all methods (upper6D dataset)

Method	Standard C	SSE2
DTW8	$\frac{1}{18}\times$	–
DTW8–Sakoe	$\frac{1}{13}\times$	–
DTW8–LB_{Keogh}	$\frac{1}{5}\times$	–
model–DTW	5.6×	–
MCS8	1×	2.1×
MCS–Partial	2.5×	2.8×
MCS–fastNN	29.1×	37.7×
MCS–DOS	221.4×	382.6×
MCS–fastKNN5	15.0×	15.0×
SVM	6.1×	6.1×

8.3 Exploring computational efficiency of fastNN

In this section, we explore computational efficiency of the tree–based fastNN algorithm, trying to offer additional insight on the future performance of our approach on unknown gesture datasets.

8.3.1 FastNN and fastKNN

As a first step, we tested our implementation for fast K–Nearest Neighbor (*fastKNN*), which is a direct generalization of fastNN (Sec. 5.1.3). Table 8.3 shows time speedups for various values of parameter K. Once again, we observe that speedup gain is higher for the upper6D dataset, which contains more training examples.

TABLE 8.3: Execution time speedup for fast KNN (Standard C)

Method	digits6D dataset	upper6D dataset
MCS8	1×	1×
K = 1	17.2×	29.1×
K = 3	11.5×	18.8×
K = 5	8.5×	14.9×
K = 7	6.3×	12.1×
K = 9	5.6×	10.9×
K = 11	4.6×	9.4×

8.3.2 Initializing fastNN and DTW

In this experiment, we tested our improved version of DTW8–LB_{Keogh}, using the training example of minimum Euclidean Distance as a good initial estimate of the minimum DTW distance example. Initialization was performed through fastNN search (Sec. 8.1.2). In our experiments, we evaluated both our approach and the ideal case, where the true global minimum DTW distance is provided as input. Obviously, the computational time required by fastNN is added to that of DTW.

Table 8.4 shows time speedups for two gesture datasets (digits6D and upper6D). As we see, for the digits6D dataset, fastNN initialization provides a speedup of around 1.47×, while it is very close to the ideal speedup (1.51×). Even more interestingly, for the large upper6D dataset, the two initialization methods provide almost same speedup.

TABLE 8.4: Execution time speedup for DTW8–LB_{Keogh}

Method	digits6D dataset	upper6D dataset
DTW8–LB_{Keogh}	1×	1×
Ideal Init	1.51×	1.39×
fastNN Init	1.47×	1.39×

We also evaluated the initialization method of fastNN search. In its original form, fastNN is initialized based on the distance of the query vector to the first leaf node reached (Depth–only search). In our experiment, we artificially posed the global minimum distance as the initial distance. We evaluated both standard and ideal fastNN by counting the number of backtrackings caused in each case. Our experiments on the digits6D and upper6D datasets revealed that ideal fastNN only saves 5% of the backtrackings, providing a speedup of 1.05×. This result shows that depth–only search stage provides an almost optimal initial candidate for the rest of the fastNN search and thus further optimization is probably meaningless.

Based on the above results, we can conclude that depth–only search and fastNN search provide almost optimal initial candidates for DTW and fastNN algorithms respectively, at least for the specific gesture datasets we used in our experiments. Thus, future research should probably focus on seeking alternative – possibly better – algorithms rather than more effective initialization steps.

8.3.3 Efficiency of fastNN in gesture spotting

In this section, we evaluate the efficiency of using *Rule2* in gesture spotting (Sec. 8.1.1). To this end, we repeated 4 of our spotting experiments, on the synthetic mixed datasets, Case I, for EasyDigits, Digits6D, Lower6D and Upper6D, keeping the results of the sliding window searches, i.e. the number of backtrackings, cosine similarity score, found gesture class and window length. We repeated these experiments twice: the first time using the same threshold values as in our standard gesture spotting experiments (column T_B in Tables 7.4 – 7.11) and the second time with this constraint removed.

By counting the total number of backtrackings required in each case, we saw that Rule2 leads to 4 times less backtrackings on average (actual values ranged from 3.7× to 4.4×). Moreover, Rule2 accepts around $5 - 7\%$ of the candidates, rejecting all others at a minimum cost of T_B backtrackings. Thus, Rule2 is indeed a meaningful approach, offering a significant computational speedup in the spotting procedure.

On the other hand, we saw that depth–only search fast Nearest Neighbor (DOS–fastNN) provides a speedup of 1.5× over Rule2–fastNN, at the cost of lower F1–score ($0 - 5\%$ drop, depending on the dataset, see Sec. 7.3.4). Summarizing partial results from previous chapters, we see that DOS–fastNN performs much worse than standard fastNN in isolated gesture recognition, but only slightly worse in the gesture verification and gesture spotting tasks, with results varying depending on the capturing device and gesture vocabularies. Additionally, it provides increased computational efficiency (1.5×), thus becoming the best method when high recognition accuracy is not that critical.

8.3.4 FastNN with varying number of training examples

It is already known that the computational gain of fastNN is higher for a larger number of training examples, M, stored in the tree structure [106, 121]. Indeed, as we saw in Sec. 8.2.3, fastNN performs much more efficiently on the upper6D dataset instead of the digits6D dataset. In this experiment, we explore this relationship in detail, producing analytical curves of execution times with respect to the number of training examples.

To this end, we varied the number of training examples per user and repeated our experiments. Please note that we used all the training users included in each dataset, to avoid bias from out–of–vocabulary examples. Fig. 8.1–a shows the per–query execution time, T, as a function of the total number of training examples, M, in a log–log plot. We observe that the relationship is captured by a straight line in the logarithmic axis, i.e. by a *power function curve* in the standard axis.

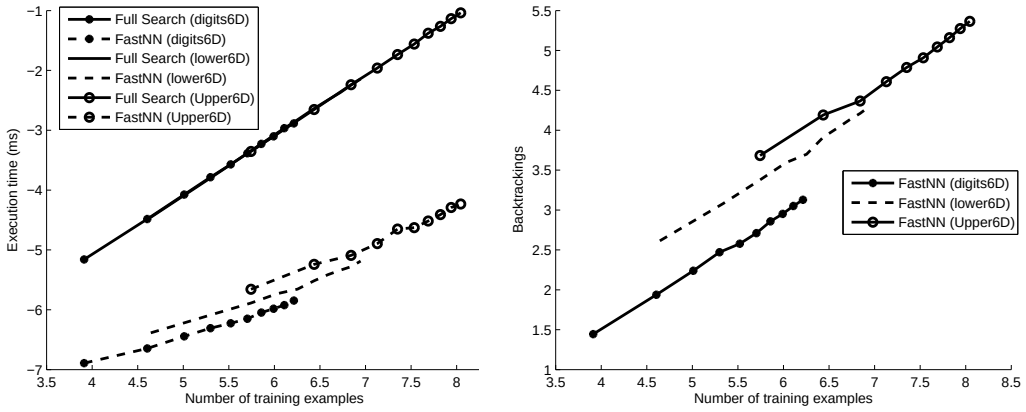
Indeed, a power function curve

$$y = a \cdot M^k \tag{8.1}$$

becomes a linear curve in the log–log plot:

$$\log y = k \cdot \log M + \log a \tag{8.2}$$

where k is the slope of the line and $\log a$ the intercept on the log–y axis.

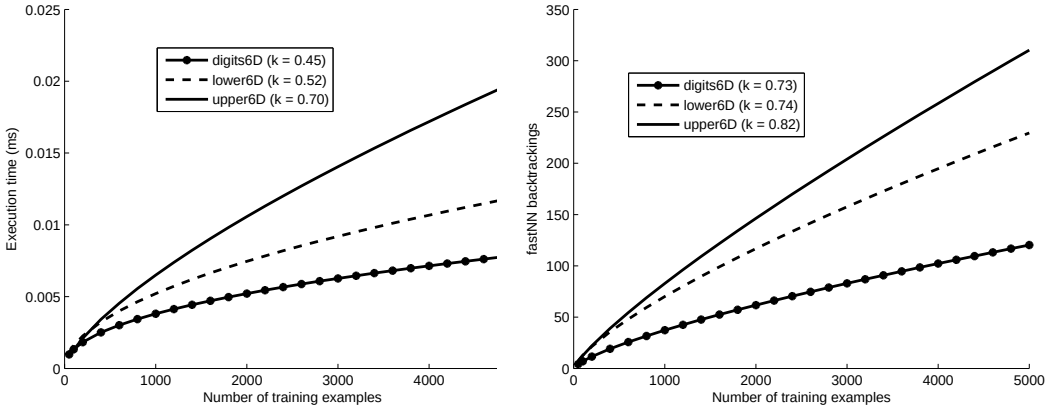


(a) (b)

FIGURE 8.1: Log–log plots, showing (a) execution time and (b) fastNN backtrackings with varying number of training examples.

As expected, the slope of FullSearch is very close 1 for all three datasets ($k \approx 0.98$ on average), since its complexity is linear to the number of training examples, M. On the other hand, the complexity of fastNN is sub–linearly related to M ($k < 1$) [106, 121]. We also note that the three FullSearch lines overlap almost perfectly to each other, eliminating any suspicion for bias due to the experimental setup. On the other hand, the fastNN lines present some variation, confirming that computational performance is related to the nature of the data [121]. Indeed, similar differences appear in the relationships between fastNN backtrackings, B, and M (Fig. 8.1–b).

To robustly estimate the slope for the fastNN lines, we used Least-Squares Estimation. We further excluded the first two pairs obtained for the upper6D dataset, as they seem



(a) (b)

FIGURE 8.2: Estimated power curves, showing measured (a) execution time and (b) fastNN backtrackings with varying number of training examples.

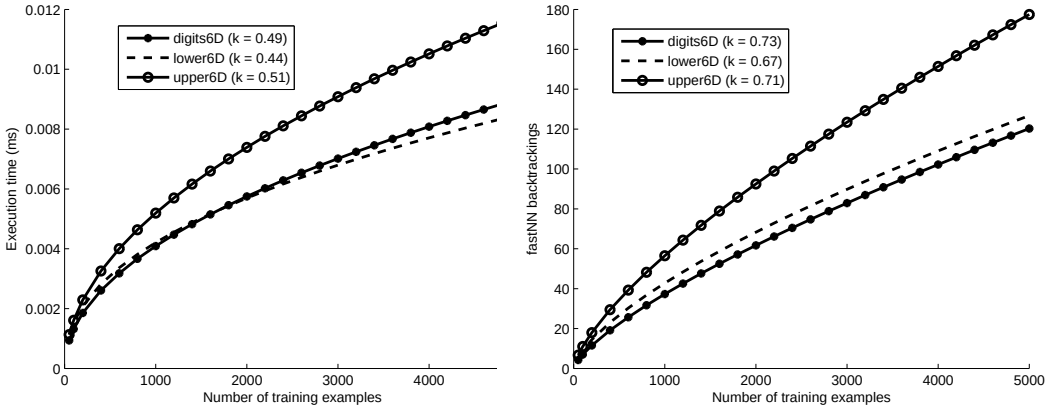
to belong to a different line. Please note that such decision rather decreases than increases bias, as the slope tends to stabilize for larger numbers of training examples. The corresponding power curves, along with the estimated slopes are shown in Fig. 8.2.

8.3.5 FastNN with varying number of vocabulary classes

While the results of previous section suggest that recognition of digits can be performed in a more efficient way, compared to lower–case and upper–case letters, the analysis ignores the different number of classes between digits and letters. For this reason, we also investigate the role of the number of gesture vocabulary classes in computational performance of fastNN.

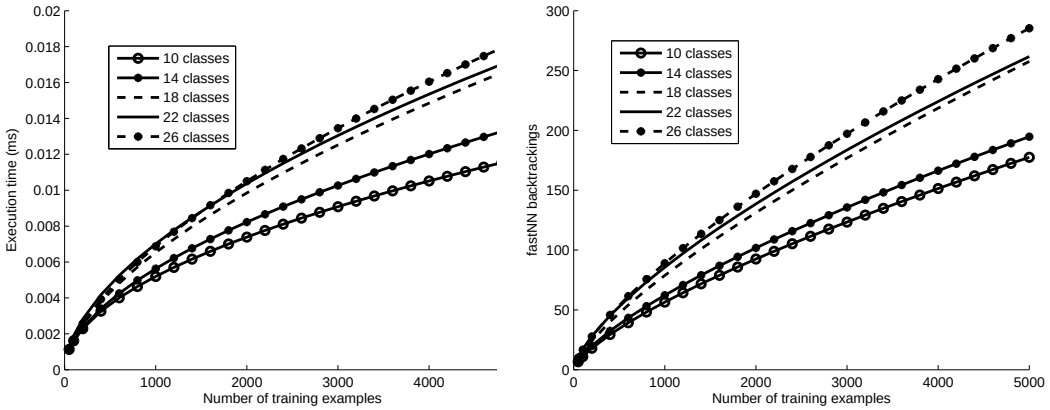
To this end, we repeated the previous experiments, using only the first 10 classes from each dataset. The resulting power curves for execution times and fastNN backtrackings are shown in Fig. 8.3. We observe that the differences between datasets are now smaller, with lower6D almost overlapping with digits6D. By comparing the number of backtrackings for 10 classes (Fig. 8.3–b) to the number of backtrackings for 26 classes (Fig. 8.2), we see that more classes (i.e. higher data diversity) result in more difficult recognition. One may also observe the importance of basing comparisons on the number of backtrackings: execution times in Fig. 8.3–a and Fig. 8.2–b differ, due to overall computing system load (CPU usage, memory usage, etc.).

Finally, we performed an additional experiment on the upper6D dataset, varying the number of classes. Fig. 8.4 show the corresponding results for some characteristic cases. One can observe that once again data diversity affects the performance of fastNN.



(a) (b)

FIGURE 8.3: Power function curves, showing estimated (a) execution time and (b) fastNN backtrackings with varying number of training examples, when only 10 classes are used from each dataset.



(a) (b)

FIGURE 8.4: Power function curves, showing estimated (a) execution time and (b) fastNN backtrackings with varying number of training examples and number of classes, on the upper6D dataset.

Based on the above results, we can conclude that recognition of digits can be performed in a more efficient way, compared to lower–case and upper–case letters, regardless of the number of vocabulary classes. The reasons for these differences are not completely revealed yet, but may be attributed to the different nature of the alphabets, since shapes of letters are much more complicated than digits.

8.4 Real–time implementation

As a final step, we developed a real–time 2D camera–based application, to further explore the effectiveness of our gesture spotting approach. FastNN tree training used 270

gestures from the original GreenDigits dataset. Our current experiments showed that hand detection tasks are much more time consuming, while gesture spotting occupies only a small portion of the execution time (5%). Further optimization, targeting low–power, mobile devices, is included in our plans for future work.

Analytically, the hotspots found in our experiments were:

- Skin detection (8%): we used the precalculated skin color histograms, as described in Sec. 4.1.3.2.
- Motion detection (6%)
- Connected component analysis (10%)
- Corner detection (13%)
- Optical flow estimation (20%)
- Morphological image processing (22%)
- Image processing & other operations (16%)
- Gesture spotting (5%)

8.5 Discussion

In this chapter, we evaluated the computational efficiency of our approach, providing a fair comparison between DTW, SVM and MCS on two isolated gesture datasets, exploring various parameters, such as effect of fastNN and SIMD instructions. Additionally, we proposed a novel method to speedup DTW, using fastNN for initialization, which resulted into 33% savings in computational time. In short, fastNN–MCS proved to be highly efficient, which allowed us to build a real–time gesture spotting application, using a standard 2D camera.

Additionally, we evaluated the performance of fastNN with varying number of training examples and gesture classes, showing that the number of fastNN backtrackings can be modelled as a power function of the number of training examples. Further investigation of this interesting result is included in our plans for the future.

Chapter 9

Hand posture recognition

Contents

In this chapter, we explore and evaluate local and global features for hand posture recognition, i.e. recognition of static hand shapes, where most of the information lies in finger configuration. Our main result regarding isolated posture recognition is that local features, such as Fourier Descriptors, can be highly informative for certain vocabularies, while global features, such as the number of fingers, can be used to improve the results through search space reduction.

Moreover, we use a lot of the techniques described in Chapters 5 – 8, confirming several important results, regarding the high computational efficiency of fast Nearest Neighbor (fastNN) and its application on posture verification, based on the number of fastNN backtrackings.

9.1 Posture recognition

Our approach assumes a reliable hand detection first step, based on depth or 2D cameras, as described in sections 4.1.2 and 4.1.3, respectively. We also assume that the posturing hand remains still for a sufficient amount of time (*posture locking phase*), in a way to trigger the system in order to proceed with feature extraction and posture recognition. Hand detection results in a binary palm mask, out of which we can extract local shape features, such as Fourier Descriptors (Sec. 4.3.2.1), or global features, based on finger characteristics (Sec. 4.3.2.2), found by our method for finger detection (Sec. 4.2). Fig. 9.1 provides the general structure of our posture recognition system.

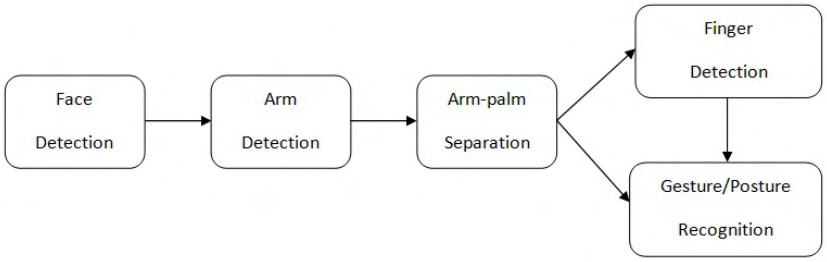


FIGURE 9.1: General structure of our proposed posture recognition system, using color and depth information. Face detection guarantees the existence of a user. Finger detection is optional, improving recognition results through search space reduction (as described in Sec. 9.2.2).

9.1.1 Nearest Neighbor search

Given a set of training examples, $U^i, i = 1, \ldots, M$, we first apply feature extraction (as described in Sec. 4.3.2) to obtain their corresponding 1D signals $\vec{u}^i$. Using the same process, we transform a test isolated posture into its corresponding query 1D signal, $\vec{q}$. We then classify $\vec{q}$ by locating the training vector $\hat{u}$ showing the Minimum Euclidean Distance (Nearest Neighbor – NN):

$$\hat{u} = \arg \min_{\vec{u^i} \in U} \|\vec{q} - \vec{u^i}\|^2 \tag{9.1}$$

and assigning its label to q.

As in the case of dynamic trajectories, we use the tree-based fast Nearest Neighbor (NN) method of Katsavounidis et al.[106] for faster search.

9.1.2 Posture verification

Posture verification accepts as valid the postures of a predefined vocabulary and rejects all other palm–finger configurations. Invalid postures include all out–of–vocabulary postures, but also intermediate versions of valid postures, appearing during posture locking.

To deal with invalid postures, we use the standard rules described in Chapter 6, i.e. *Rule1*, based on Euclidean Distance, and *Rule2*, based on the number of fastNN backtrackings. Algorithm 2 summarizes the main steps of the unified posture recognition and verification process.

Algorithm 2 Isolated posture recognition with rejection of invalid examples

Input: hand coordinates x, y
thresholds T_B, T_D
Output: Label C of the gesture

1: Represent posture as described in Sec. 4.3.2
2: $(B, d, c) \leftarrow$ fastNNsearch(x, y, N) ▷ *B: # backtrackings, d: Euclidean distance, c: predicted class label*
3: $C \leftarrow$ CheckRule3(B, d, T_B, TD, c) ▷ Alternatively one can use Rule1 or Rule2
4: **return** C ▷ C is the final recognition result

CheckRule1: Gesture is valid if $d < T_D$
CheckRule2: Gesture is valid if $B < T_B$
CheckRule3: Gesture is valid if $B < T_B$ AND $d < T_D$

9.1.3 Posture locking

Posture spotting is typically achieved through holding the hand still on the air (*posture locking phase*). In our work, we posed this assumption in a 2D camera system, performing hand detection using the method of Sec. 4.1.3, i.e. face detection, skin detection and corner detection. When the variance of hand position over T_s frames was smaller than a threshold T_v, we further performed feature extraction ($FD(z)$) and posture recognition. In our implementation, we used $T_v = 5$ pixels and $T_s = 30$ frames, i.e. 1 second in a VGA camera–30 fps system. However, values of T_v, T_s are application–dependent and can be changed accordingly, to better fit a specific environment or user category.

9.2 Experimental results

9.2.1 Experimental setup

For our experiments, we used the dataset of Ren et al.[13, 128], which contains an alphabet of 10 different postures, with 10 examples from 10 different persons, i.e. 1000 postures in total (Fig. 3.3). As described in Sec. 3.2.2, the original work of Ren et al.[13, 128] required the user to wear a black bracelet on the gesturing hand's wrist, for easier hand–palm separation. In our work, we avoid the use of a black bracelet, using color information only for face detection during initialization and depth information for all the other tasks, i.e. hand detection and segmentation (Sec. 4.1.2), finger detection (Sec. 4.2) and posture representation (Sec. 4.3.2).

In total, we evaluated the following five methods:

- $FD(z)$: Fourier Descriptor using the complex contour signal, $z(n)$ (Eq. 4.7)
- $FD(r)$: Fourier Descriptor using the signal of distances of contour points from palm centroid, $r(n)$, (Eq. 4.5)
- $FD^*(z)$: $FD(z)$, using the number of fingers for search space reduction
- $FD^*(r)$: $FD(r)$, using the number of fingers for search space reduction
- *Fingers*: Global representation, using finger characteristics (height, width) and their relative distance from the leftmost finger (Sec. 4.3.2.2)

9.2.2 Isolated recognition

For our isolated recognition experiments, we used 10–fold cross-validation in a user–independent mode, i.e. at each round we used 900 postures (9 people $\times$ 10 categories $\times$ 10 examples) for training and the rest 100 examples for testing.

Our first experiment explores the role of the Fourier Descriptor parameter P on the performance of the $FD(z), FD(r), FD^*(z)$ and $FD^*(r)$ methods. Please note that methods $FD(r)$ and $FD^*(r)$ use half the number of features, compared to methods $FD(z)$ and $FD^*(z)$, for fixed P. Fig. 9.2 shows recognition accuracies for various values of P. We observe that the hybrid methods ($FD*(z)$ and $FD^*(r)$) outperform the purely local methods ($FD(z)$ and $FD(r)$), even for smaller P. Moreover, while $FD(r)$ performs 1% worse than $FD(z)$, this difference reduces to 0.4% for $FD^*(z)$ and $FD^*(r)$. The higher accuracy of the hybrid methods may be explained by the very high accuracy

on the number of fingers detected (996/1000). Based on the above results, we choose $P = 8$ for all four methods, as a good compromise between recognition accuracy and computatioanl efficiency.

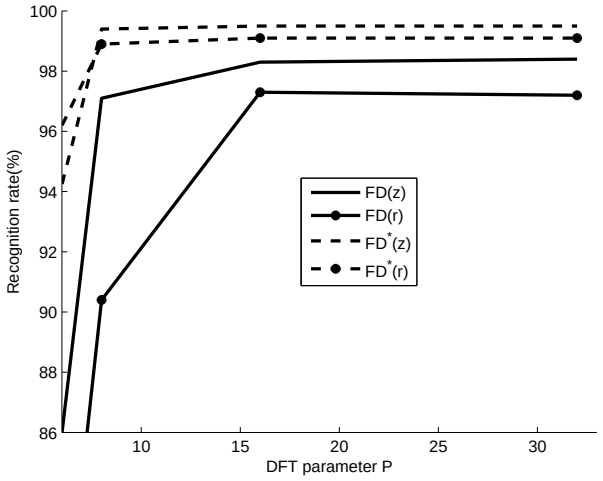


FIGURE 9.2: Recognition accuracy of the four Fourier Descriptor methods, for various values of parameter P. Please note that methods $FD(r)$ and $FD^*(r)$ use half the number of points, compared to methods $FD(z)$ and $FD^*(z)$.

Table 9.1 shows the best results achieved by each method, along with the results of our global features (*Fingers*) and the global approach of Ren et al.[13, 128]. We observe that our global features (*Fingers*) proved better than [13, 128] but not as good as the four FD–based methods. Overall, these results suggest that local features, such as Fourier Descriptors, can be very informative and sufficient for hand posture recognition, at least for the specific vocabulary used in our experiments. On the other hand, global features, such as the number of fingers, can be used to improve the results through search space reduction. This is in contrast to Ren et al.[13, 128], who proposed and compared only global methods, assuming that they are more robust than local features for posture recognition.

Method	Rec. Accuracy (%)
$FD(z)$	98.4%
$FD(r)$	97.3%
$FD^*(z)$	99.5%
$FD^*(r)$	99.1%
Fingers	96.3%
Ren et al.[13, 128]	93.9%

TABLE 9.1: Recognition accuracy for various methods. The FD^* methods use Fourier Descriptors and achieve search space reduction based on the number of fingers.

Table 9.2 shows the confusion matrix for the $FD(z)$ method, which provides high recognition accuracy (98.4%), without the need for finger detection. Assuming a uniform prior

distribution on the 10 postures, initial entropy over gesture categories is reduced from $\log_2 10 = 3.3219$ bits to 0.1133 bits. The corresponding normalized mutual information (Eq. 5.13) was found to be 96.59%.

	a(0)	b(1)	c(3)	d(2)	e(2)	f(2)	g(4)	h(5)	i(1)	j(2)
a(0)	1									
b(1)		0.99	0.01							
c(3)			1							
d(2)				1						
e(2)				0.05	0.94					0.01
f(2)				0.01		0.99				
g(4)							0.97		0.03	
h(5)			0.01					0.97	0.01	0.01
i(1)								0.01	0.99	
j(2)								0.01		0.99

TABLE 9.2: Confusion matrix for the $FD(z)$ method, showing probabilities of misclassification among the various classes. Posture naming follows the order used in Fig. 3.3, while the numbers in parentheses indicate the number of fingers

Our second experiment involves varying the number of training users, in a leave–K–users–out cross–validation scheme. Specifically, we considered K users for training set and $10 - K$ users for validation, and averaged the results over 100 random rounds. As we observe in Fig. 9.3, recognition accuracies follow the same trend as in Table 9.1, with $FD^*(z)$ being the best method and *Fingers* the least preferred. Quite interestingly, the two hybrid approaches are almost invariant to the number of training users, showing high results even for 1 user. Once again we see that finger detection may lead to much higher recognition accuracies, especially for small posture vocabularies, as in our case (10 postures).

9.2.3 Posture verification

We also evaluated the posture verification component of our system, based on *Rule1* (Euclidean Distance) and *Rule2* (number of fastNN backtrackings), as described in algorithm 2. To this end, we considered 5 postures as positive and 5 as negative classes. We used 2 users to build the fastNN tree, and 2 distinct users for each of the other four subsets (positives/negatives × validation/testing). Finally, we produced the Receiver Operating Characteristic (ROC) curves, showing the trade–off between Precision and Recall for each verification rule.

The verification results may vary, according to the split of classes and the separability of positive to negative examples. Fig. 9.4–a shows the ROC curves for a random split,

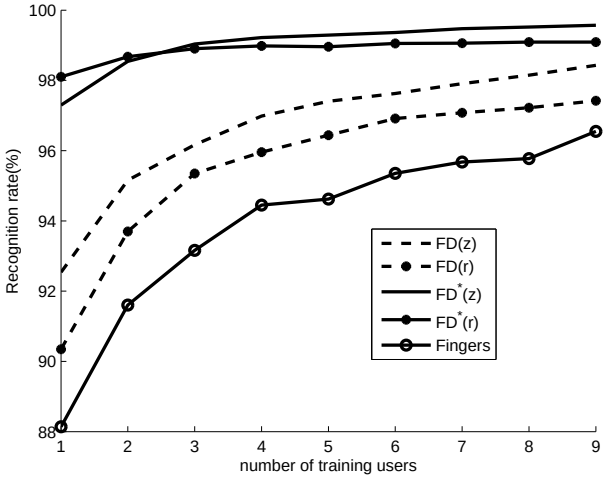
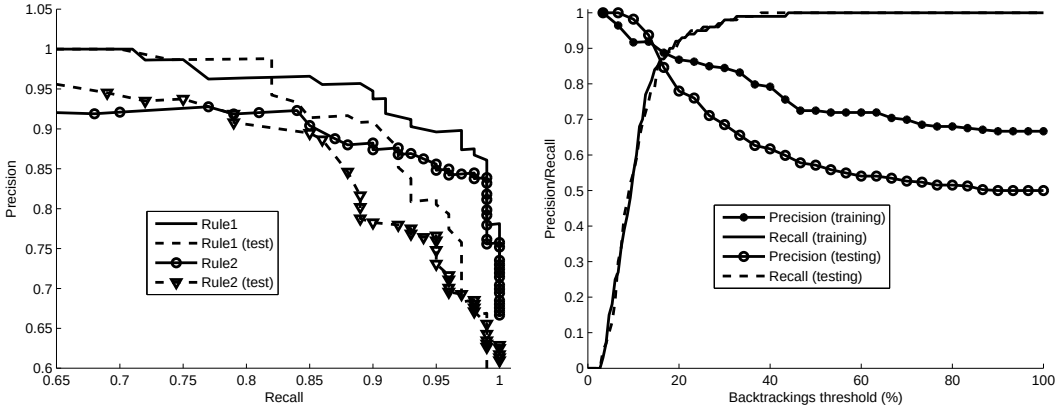


FIGURE 9.3: Recognition accuracy of the five posture recognition methods with varying number of training examples.



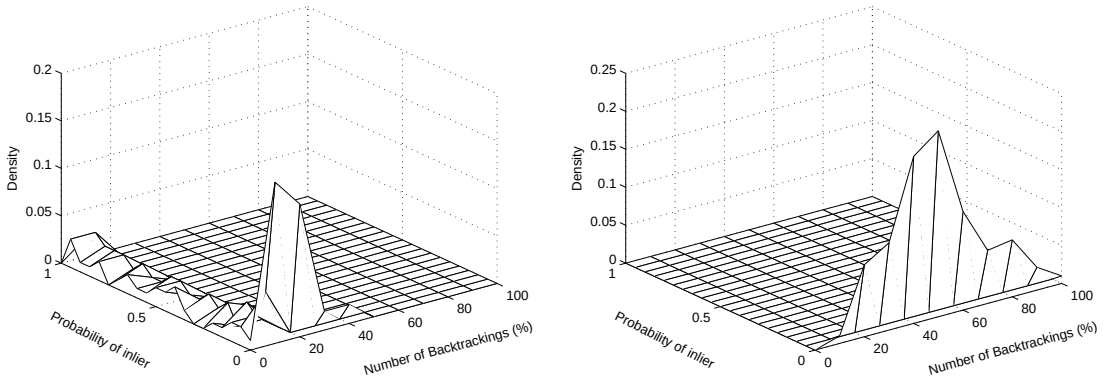
(a) (b)

FIGURE 9.4: (a)Precision–Recall trade–off for rejection of 5 out–of–vocabulary posture classes by a system trained on 5 different postures. (b) Effect of thresholding the number of backtrackings at various values. The EER points can be achieved for a low threshold ($T_B \approx 20$), resulting in increased computational efficiency.

chosen such that it corresponds to the *average case*, i.e. not to the best or the worst. However, it is worth noting that we didn't observe big differences among curves in other random splits (around 1% difference in the EER point). Fig. 9.4–b also shows how Precision and Recall vary for different values of the backtrackings threshold, T_B. We observe that while Recall remains almost unchanged between training and testing sets, Precision differs due to existence of different postures. The EER points can be achieved for a low threshold ($T_B < 20\%$ of the training examples), resulting in increased computational efficiency.

We repeated this experiment 100 times, each with different random splits of users and classes. Fig. 9.5 shows the resulting joint distributions of inlier probability and fastNN

backtrackings, $(P0, B)$, where we observe that invalid gestures show a lower probability P_0, while they cause a larger number of backtrackings B, in general. On the other hand, valid gestures show higher P_0, while B almost never exceeds 40% of the number of fastNN tree examples. We also observe that B is lower for higher P_0, as expected. The above results agree with the results presented in Chapter 6 for dynamic trajectories.



(a) (b)

FIGURE 9.5: Joint distribution of inlier probability and fastNN backtrackings, (P_0, B), for (a) valid and (b) out–of–vocabulary gestures, on a system trained on 5 postures, for 100 random trials. B is shown as a percentage over the number of fastNN tree examples.

9.2.4 Computational efficiency

As a final step, we measured the computational efficiency of our Nearest Neighbor–based approach for hand posture recognition. Our experiments were run on a typical desktop environment, consisting of an Intel(R) Core(TM) i7 CPU running Microsoft (R) Windows (TM) 7 64–bit OS. We implemented all methods in C, applying some basic code optimization and compiled them using Microsoft Visual Studio 2010, Release configuration, *-Od* compiler–optimized executable, to make results as platform–independent as possible. Wherever applicable, we optimized our code by using Single–Instruction–Multiple-Data (SIMD) techniques, namely using the "SSE2" enhanced instruction set supported by most modern Intel CPU IA-32 and Intel-64 architectures.

Table 9.3 shows speed–ups for four Nearest Neighbor variations for the $FD(z)$ method, under both standard–C and SSE2 implementations. We normalized results using full-search Nearest Neighbor as point of reference. We observe that fastNN speeds up the search process by 20× under standard C implementation and 31.5× when SIMD–SSE2 intrinsics are used.

We also observe a very important speedup of 77× and 131× when the depth–only version of fastNN (DOS–NN) is used, under standard C and SSE2 implementations respectively.

TABLE 9.3: Execution time speedup for 4 NN variations of the $FD(z)$ method

Method	Standard C	SSE2
NN–Full Search	1×	2.1×
NN–Partial	1.9×	2.4×
NN–fastNN	19.9×	31.5×
NN–DOS	77.4×	131.3×

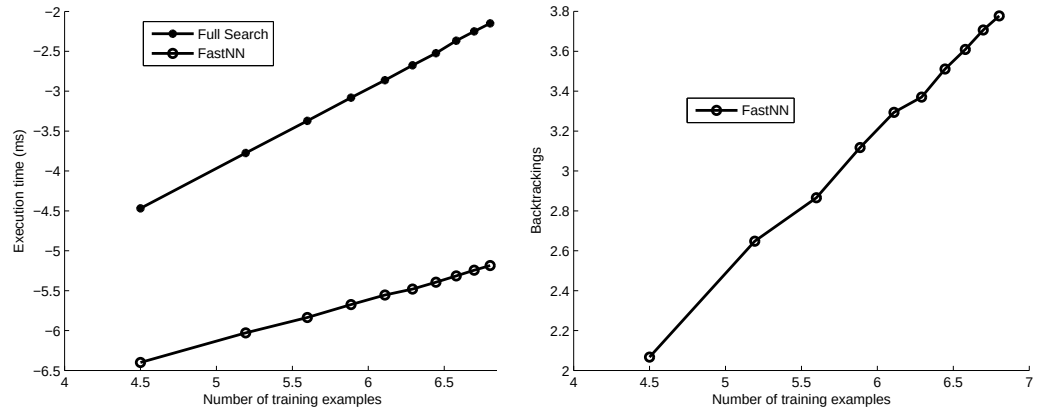
Quite interestingly, DOS–NN achieved 96.4% recognition accuracy, i.e. only 2% lower than standard $FD(z)$. Thus, high recognition accuracy may be achieved along with significant computational speedup.

Finally, we explored the performance of fastNN with varying number of training examples, as we did for dynamic trajectories in Sec. 8.3.4. To this end, we varied the number of training examples per user and repeated our experiments, using all training users included in the dataset to avoid bias from out–of–vocabulary examples.

Fig. 9.6–a shows the per–query execution time, T, as a function of the total number of training examples, M, in a log–log plot. We observe that the relationship is captured by a straight line in the logarithmic axis, i.e. by a *power curve* in the standard axis. As in Sec. 8.3.4, we estimated the slopes of the log–log curves using Least-Squares Estimation and modelled them as power function curves (Eq. 8.1). As expected, the slope of FullSearch was very close 1, implying complexity linear to the number of training examples, M. On the other hand, the complexity of fastNN is sub–linearly related to M, with slope close to 0.52 [106, 121]. As a comparison, the execution time slopes for the three dynamic trajectory datasets were found to be approximately 0.45 for digits6D, 0.52 for lower6D and 0.70 for upper6D.

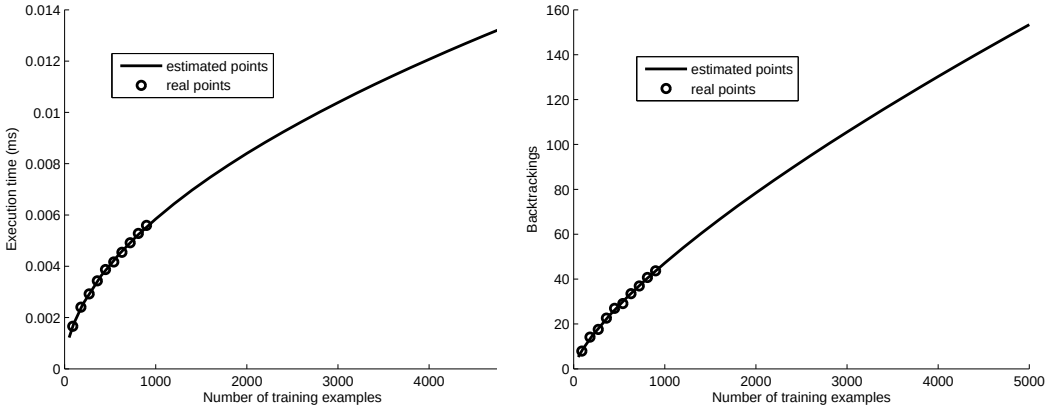
Fig. 9.6–b shows the per–query number of backtrackings, T, as a function of the total number of training examples, M, in a log–log plot. The estimated slope us $k = 0.73$, which is comparable to the slopes estimated for the digits6D ($k = 0.73$), lower6D ($k = 0.74$) and upper6D ($k = 0.82$) datasets.

Figs. 9.7 a–b show the corresponding power function curves for execution time and fastNN backtrackings. We observe that 50 backtrackings are required on average when 1000 training examples are used, explaining the 20× speedup of fastNN (Table 9.3). We also see the nice fitting of measured points to the estimated power curve, proving that such analysis is quite meaningful and may generalize well on other parameter values.



(a) (b)

FIGURE 9.6: Log–log plots, showing (a) execution times and (b) fastNN backtrackings with varying number of training examples.



(a) (b)

FIGURE 9.7: Power function curves, showing estimated (a) execution time and (b) fastNN backtrackings with varying number of training examples.

9.3 Discussion

In this chapter, we evaluated our approach for hand posture recognition, based on local and global features. Our experiments explored several aspects of the problem, such as recognition with varying number of training examples, posture verification, real–time posture locking and computational efficiency of fast Nearest Neighbor search (fastNN).

Our main result regarding isolated posture recognition is that local features, such as Fourier Descriptors, can be highly informative for 10 simple hand shapes, achieving state-of-the-art results on a challenging dataset. On the other hand, global features, such as the number of fingers, can be used to improve the results through search space reduction. This is in contrast to previous approaches [13, 128], which assumed that global methods are always more robust for posture recognition.

Regarding posture verification, we showed that Rule2, based on the number of fastNN backtrackings (B), can be very effective in filtering out the majority of invalid examples. More importantly, regarding the relationship between B and the inlier probability, $P0$, we observed the same patterns as in recognition of dynamic trajectories, proving that computational performance of fastNN is related to the number and nature of the training and testing examples.

Our goals for future work include formal evaluation of our posture locking method on an appropriately created dataset, where continuous recognition techniques may be applied, such as sliding windows and max–voting decisions. We also intend performing experiments on other datasets and exploring methods of fusing depth and color information.

Part IV

Conclusions

Chapter 10

Conclusions and Future work

Contents

10.1 Conclusions

In this thesis, we studied the problem of hand gesture recognition on continuous streams of digits and letters, exploring various trade–offs between recognition accuracy and computational efficiency on four fundamental tasks: hand detection and feature extraction, isolated recognition, gesture verification, and gesture spotting on continuous data streams.

Specifically, in Chapter 4, we presented our approach for hand detection and a novel technique for finger segmentation for various data acquisition sources. Moreover, we presented our approach for gesture representation, targeting translation, rotation and scaling invariance. Additionally, our approach is independent of the type of capturing device.

In Chapter 5, we studied isolated recognition for trajectories of digits and letters. Our approach is based on Maximum Cosine Similarity (MCS) and a tree–based fast Nearest Neighbor algorithm. In our experiments, conducted on three publicly available databases, we explored various parameters, such as recognition with a varying number of training users/examples, performance on noisy data and common recognition of digits and letters. Chapter 5 confirmed the high recognition accuracy achieved by MCS, making it a promising method for gesture spotting, as well.

In Chapter 6, we studied the problem of gesture verification and presented our approach for rejection of invalid gesture instances, based on thresholding the cosine similarity score and the number of fastNN backtrackings required to classify an unknown gesture vector. Our approach can effectively reject three main types of invalid gestures: *out–of–vocabulary* gestures, *noisy* gestures and *random* movements. We additionally showed that there exists a non–linear relationship among number of backtrackings, cosine similarity score and the probability of inlier. This property leads to high computational efficiency, as we may constrain the search time up to a certain number of backtrackings, rejecting all other inputs as invalid gestures. Finally, we showed an interesting relationship between computational efficiency and classification accuracy, through the Minimum Backtrackings Classifier, which may prove quite promising in the future.

In Chapter 7, we proposed a complete gesture spotting framework for continuous streams of digit and letter trajectories in an *indirect* way, i.e. without using motion cues or other marks to detect the starting and ending time boundaries of the performed gestures. Our approach is based on applying isolated classification through fastNN Maximum Cosine Similarity (fastNN–MCS), collecting groups of overlapping gesture candidates and applying conflict resolution methods to reach the final spotting result. Our main contribution lies in proposing a novel method to perform conflict resolution, combining cosine similarity score and time duration of gesture candidates. Moreover, we introduced a probabilistic framework to learn and handle subgesture relationships, extending previous models which used strict rules for the gesture classes. Specifically, our approach models both categories and relative time boundaries of the gesture candidates. Finally, we also experimented with gesture spotting on real texts and designed a special version of the Weighted Edit Distance for word recognition, supporting small vocabularies of 10-40 words.

In Chapter 8, we evaluated the computational efficiency of our approach and demonstrated its low cost compared to Dynamic Time Warping (DTW). Additionally, we proposed a novel method to speedup DTW, using fastNN for initialization, which resulted into 33% savings in computational time. In short, fastNN–MCS proved to be highly efficient, which allowed us to build a real–time gesture spotting application, using a standard 2D camera. Additionally, we evaluated the performance of fastNN with varying number of training examples and gesture classes, showing that the number of fastNN backtrackings can be modelled as a power function of the number of training examples. Further investigation of this interesting result is included in our plans for the future.

Finally, in Chapter 9, we presented and evaluated our approach for hand posture recognition, based on local and global features. Our experiments explored several aspects

of the problem, such as recognition with varying number of training examples, posture verification, real–time posture locking an computational efficiency of fast Nearest Neighbor (fastNN). Our main result regarding isolated posture recognition is that local features, such as Fourier Descriptors, can be highly informative for 10 simple hand shapes, achieving state-of-the-art results on a challenging dataset. On the other hand, global features, such as the number of fingers, can be used to improve results through search space reduction.

10.2 Future work

Our goals for future work include further research on hand and finger detection methods, as well as evaluating them on new, larger datasets of postures and gestures. Additionally, we intend exploring methods of fusing depth and color information.

We also intend to further investigate gesture spotting methods, trying alternative classifiers, such as Dynamic Time Warping (DTW), Hidden Markov Models (HMM) and Conditional Random Fields (CRF), hoping that they may improve recognition results without compromising computational performance. Another promising topic in gesture spotting is that of conflict resolution, which seems to be the key concept of all indirect spotting approaches.

Seeking relationships between computational efficiency, classification and verification, attempting to relate the number of fastNN backtrackings and traditional pattern recognition quantities, such as Euclidean and Mahalanobis distance, data dimensionality and number of training examples and classes, is always a fascinating problem, although not a trivial one, mainly due to its highly stochastic nature. Further investigation of such relationship, including datasets from other domains, such as face and activity recognition, is certainly included in our future plans.

Finally, we plan to develop more efficient hand and finger detection methods, and evaluate them on large datasets of postures and gestures, as well as through subjective evaluations on real users. We also plan to perform a formal evaluation of our posture locking method on an appropriately created dataset, where continuous recognition techniques may be applied, such as sliding windows and max–voting decisions. We also intend performing experiments on other datasets and exploring methods of fusing depth and color information.

Appendix A

Fast Nearest Neighbor algorithms

A.1 The Nearest Neighbor problem

Given a set of training vectors, $U = \{u^i \in \mathbb{R}^d, i = 1, \ldots, M\}$, and a query vector, $q \in \mathbb{R}^d$, the problem of Nearest Neighbor consists of locating the training vector $\hat{u}$ which shows the minimum distance, $d(q, u^i)$:

$$\hat{u} = \arg \min_{\vec{u^i} \in U} d(q, u^i) \tag{A.1}$$

In this work, we used Euclidean distance between two vectors q, u:

$$d(q, u) = \|q - u\|^2 \tag{A.2}$$

A.2 Exact Nearest Neighbor algorithms

The obvious brute–force algorithm (*Full Search*) is to compute all distances in a linear way, showing computational complexity which is linear to the number of training examples, M, and the data dimensionality, d, i.e. $\mathcal{O}(M \cdot d)$. *Partial Distance Search* (PDS) improves the Full Search algorithm through early termination of local distance computation when it exceeds the running minimum distance [127], but its cost remains nearly linear.

Besides PDS, a lot of fast Nearest Neighbor (fastNN) methods have been proposed in the literature [106, 129–133], achieving nearly logarithmic time on average. Such algorithms are typically based on partition trees, arranging training vectors in a tree data structure in some appropriate way (*initialization step*). Initialization typically starts from the

root of the tree and distributes training vectors across its b children based on $b-1$ hyperplanes; it then continues recursively for all children nodes. During the *searching step*, fastNN algorithms navigate the tree recursively until the first leaf node is reached; at this point, a first solution is acquired (*Depth–Only Stage* – DOS). At that point, backtracking to previous tree nodes is performed, examining the rest of the training vectors. Computational efficiency comes from pruning many training vectors out of the search based on effective lower bounds.

Partition tree–based algorithms vary based on the selection method of hyperplanes and lower bounds. K–d tree [129, 130] was one of the first such methods, using hyperplanes perpendicular to the coordinate axes, partitioning the search space into hyper–rectangular regions (*buckets*), each containing a small number of training vectors [131]. K-d tree searching involves reaching a first candidate and setting a *query hypersphere* which bounds the global minimum distance; it then searches exhaustively all buckets which intersect with that hypersphere. Performance of k–d trees is typically compromised at higher dimensions, since many buckets tend to intersect with the query hypersphere. Guttman [132] proposed R–trees, which partitions training vectors based on hyper–rectangular regions (instead of hyperplanes), showing performance similar to k–d trees for higher data dimensions.

Katsavounidis et al.[106] proposed a fastNN algorithm which partitions data based on recursive applications of the k–means clustering method, while it uses a lower bound based on projections of vectors on a maximum separating hyperplane. Since this is the main fastNN algorithm used in this thesis, we described it in detail in Sec. 5.1.2. McNames [133] improved the above algorithm by separating training vectors using their projections on the principal axis of the training set (*Principal Axis Tree* – PAT). During searching, a lower bound was used for pruning, derived by the law of cosines for triangles. In his experiments [121], McNames compared 17 fastNN algorithms and showed that PAT and the fastNN algorithm of [106] outperformed all other algorithms, while their relative performance varied based on the type of training data.

A.3 Approximate Nearest Neighbor algorithms

While exact search can be efficiently performed when the data dimensionality, d, is quite low, it may be prohibitively expensive for large d values. For this reason, a lot of approximate fastNN algorithms have been proposed, each balancing between computational efficiency, memory requirements and quality of search. Approximate algorithms can be

generally classified into three categories [134]: (1) partitioning trees, (2) hashing techniques and (3) graph–based techniques. In the following, we describe the most important details of each category. For more information, the reader may refer to [131, 134].

Partition tree–based algorithms offer a depth–only search stage (DOS), which allows for early termination of the search process. Alternatively, one may threshold the number of backtrackings or the computational time during search. We explored this idea in detail in Chapter 6, where we studied the trade–offs between recognition accuracy and computational efficiency.

Locality Sensitive Hashing (LSH) [135] is a method to perform dimensionality reduction, mapping similar vectors to the same buckets with high probability. Since performance of LSH is highly dependent on the quality of the hashing functions used, many modifications and improvements have been proposed, such as parameter sensitive hashing [136], kernelized LSH [137] and optimized kernel hashing [138]. While hashing techniques have been successfully applied on many problems, Muja et al.[134] showed that they are typically outperformed by the DOS variant of the most efficient partition tree-based algorithms.

Graph–based techniques consider training vectors as the vertices of a graph structure, while edges connect each vector to its nearest neighbor(s). The search procedure simultes hill climbing, starting from a random initial point and moving towards the query vector [139].

Bibliography

[1] S. Mitra and T. Acharya. Gesture Recognition: A Survey. *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(3):311–324, 2007.

[2] J. Alon, V. Athitsos, Quan Yuan, and S. Sclaroff. A unified framework for gesture recognition and spatiotemporal gesture segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, 31(9):1685–1699, 2009.

[3] M. Chen, G. AlRegib, and B. Juang. 6DMG: a new 6D Motion Gesture Database. In *2nd ACM Multimedia Systems Conf.*, pages 83–88, 2012.

[4] A. Pentland. Perceptual user interfaces: perceptual intelligence. *Communications of the ACM*, 43(3):35–44, 2000.

[5] S.C.W. Ong and S. Ranganath. Automatic sign language analysis: a survey and the future beyond lexical meaning. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(6):873–891, 2005.

[6] J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *20th annual ACM symposium on User Interface Software and Technology*, pages 159–168, 2007.

[7] L. Anthony and J. O. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Proc. of Graphics Interface*, pages 245–252, 2010.

[8] Y. Li. Protractor: A fast and accurate gesture recognizer. In *28th Int'l Conf on Human factors in Computing Systems*, pages 2169–2172, 2010.

[9] L. Anthony and J. O. Wobbrock. $n-protractor: a fast and accurate multistroke recognizer. In *Proc. of Graphics Interface*, pages 117–120, 2012.

[10] D. Frolova, H. Stern, and S. Berman. Most Probable Longest Common Subsequence for recognition of gesture character input. *IEEE Trans. on Cybernetics*, 43 (3):871–880, 2013.

[11] H. Stern, M. Shmueli, and S. Berman. Most discriminating segment–Longest common subsequence (MDSLCS) algorithm for dynamic hand gesture classification. *Pattern Recognition Letters*, 34(15):1980–1989, 2013.

[12] I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *British Machine Vision Conf. (BMVC)*, pages 101.1–101.11, 2011.

[13] Z. Ren, J. Yuan, and Z. Zhang. Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In *ACM Multimedia (ACM MM 11)*, pages 1093–1096, 2011.

[14] P. Doliotis, V. Athitsos, D. Kosmopoulos, and S. Perantonis. Hand shape and 3d pose estimation using depth data from a single cluttered frame. In *Int'l Symposium on Visual Computing (ISVC)*, volume 1, pages 148–158. IEEE, 2012.

[15] A.D. Bagdanov, A. Del Bimbo, L. Seidenari, and L. Usai. Real–time hand status recognition from RGB–D imagery. In *21st Intl Conf. on Pattern Recognition (ICPR)*, pages 2456–2459, 2012.

[16] C. Keskin, F. Kirac, Y.E. Kara, and L. Akarun. Randomized decision forests for static and dynamic hand shape classification. In *IEEE Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 31–36, 2012.

[17] A. Kulshreshth, C. Zorn, and J. LaViola. Real-time markerless kinect based finger tracking and hand gesture recognition for hci. In *Proc. IEEE Symposium on 3D User Interfaces*, volume 1, pages 187–188, 2013.

[18] I. Guyon, V. Athitsos, P. Jangyodsuk, B. Hamner, and H.J. Escalante. Chalearn gesture challenge: Design and first results. In *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1–6, 2012.

[19] S. Escalera, J. Gonzàlez, X. Baró, M. Reyes, O. Lopes, I. Guyon, V. Athitsos, and H. Escalante. Multi-modal gesture recognition challenge 2013: Dataset and results. In *15th ACM on Int'l Conf. on Multimodal Interaction (ICMI'13)*, pages 445–452, 2013.

[20] Opencv library. link: http://opencv.org/.

[21] J. Suarez and R.R. Murphy. Hand gesture recognition with depth images: A review. In *IEEE Int'l Symposium on Robot and Human Interactive Communication*, pages 411–417, 2012.

[22] J. Han, L. Shao, D. Xu, and J. Shotton. Enhanced computer vision with Microsoft Kinect sensor: A review. *IEEE Trans. on Cybernetics*, 43(5):1318–1334, 2013.

[23] R. Xu, S. Zhou, and W. J. Li. MEMS accelerometer based nonspecific-user hand gesture recognition. *IEEESensors Journal*, 12(5):1166–1173, May 2012.

[24] S. Zhou, Fei Fei, G. Zhang, J. D. Mai, Y. Liu, J. Y. J. Liou, and W. J. Li. 2D human gesture tracking and recognition by the fusion of MEMS inertial and vision sensors. *IEEE Sensors Journal*, 14(4):1160–1170, April 2014.

[25] V.E. Kosmidou and L.J. Hadjileontiadis. Sign language recognition using intrinsic-mode sample entropy on sEMG and accelerometer data. *IEEE Trans. on Biomedical Engineering*, 56(12):2879–2890, 2009.

[26] V. E. Kosmidou, P. C. Petrantonakis, and L. J. Hadjileontiadis. Enhanced sign language recognition using weighted intrinsic–mode entropy and signer's level of deafness. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, 41(6):1531–1543, 2011.

[27] Y. Li, X. Chen, X. Zhang, K. Wang, and Z.J. Wang. A sign-component-based framework for chinese sign language recognition using accelerometer and sEMG data. *IEEE Trans. on Biomedical Engineering*, 59(10):2695–2704, 2012.

[28] S. Berman and H. Stern. Sensors for gesture recognition systems. *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(3):277–290, 2012.

[29] S. Kratz and M. Rohs. Protractor3D: a closed-form solution to rotation-invariant 3D gestures. In *16th Int'l Conf. Intelligent User Interfaces (IUI'11)*, pages 371–374, 2011.

[30] M. Chen, G. AlRegib, and B. H. Juang. Feature processing and modeling for 6D motion gesture recognition. *IEEE Trans. on Multimedia*, 15(3):561–571, 2013.

[31] X. Zhang, X. Chen, Y. Li, V. Lantz, K. Wang, and J. Yang. A framework for hand gesture recognition based on accelerometer and EMG sensors. *IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans*, 41(6):1064–1076, 2011.

[32] Z. Lu, X. Chen, Q. Li, X. Zhang, and P. Zhou. A hand gesture recognition framework and wearable gesture–based interaction prototype for mobile devices. *IEEE Trans. on Human–Machine Systems*, 44(2):293–299, 2014.

[33] W. Shen, K. Deng, X. Bai, T. Leyvand, B. Guo, and Z. Tu. Exemplar–based human action pose correction. *IEEE Trans. on Cybernetics*, 44(7):1053–1066, 2014.

[34] 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 *16th Int'l Conf. on Pattern Recognition*, volume 2, pages 515–519, 2002.

[35] Z. Moghaddam and M. Piccardi. Training initialization of hidden Markov models in human action recognition. *IEEE Trans. on Automation Science and Engineering*, 11(2):394–408, 2014.

[36] H. D. Yang, S. Sclaroff, and S. W. Lee. Sign language spotting with a threshold model based on Conditional Random Fields. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 31(7):1264–1277, 2009.

[37] M. Elmezain, A. Al-Hamadi, and B. Michaelis. A robust method for hand gesture segmentation and recognition using forward spotting scheme in Conditional Random Fields. In *20th Int'l Conf. Pattern Recognition (ICPR)*, pages 3850–3853, 2010.

[38] M.R. Malgireddy, J.J. Corso, S. Setlur, V. Govindaraju, and D. Mandalapu. A framework for hand gesture recognition and spotting using sub-gesture modeling. In *20th Int'l Conf. on Pattern Recognition (ICPR)*, pages 3780–3783, 2010.

[39] H. I. Suk, B. K. Sin, and S. W. Lee. Hand gesture recognition based on dynamic bayesian network framework. *Pattern Recognition*, 43(9):3059 – 3072, 2010.

[40] J. Nagi, F. Ducatelle, G.A Di Caro, D. Ciresan, U. Meier, A Giusti, F. Nagi, J. Schmidhuber, and L.M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *IEEE Int'l Conf. on Signal and Image Processing Applications*, pages 342–347, 2011.

[41] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. on Information Theory*, 13(2):260–269, 1967.

[42] R. D. Vatavu. The impact of motion dimensionality and bit cardinality on the design of 3D gesture recognizers. *Elsevier Int'l Journal of Human-Computer Studies*, 71(4):387–409, 2013.

[43] A. Hernandez-Vela, M.A Bautista, X. Perez-Sala, V. Ponce, X. Baro, O. Pujol, C. Angulo, and S. Escalera. BoVDW: Bag-of-Visual-and-Depth-Words for gesture recognition. In *21st Int'l Conf. on Pattern Recognition (ICPR)*, pages 449–452, 2012.

[44] D. Wu, F. Zhu, and L. Shao. One shot learning gesture recognition from rgbd images. In *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 7–12, 2012.

[45] C. Cortes and V. Vapnik. Support–vector networks. *Machine learning*, 20(3): 273–297, 1995.

[46] Li Deng, G. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: an overview. In *IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8599–8603, 2013.

[47] R.C. Gonzalez and R.E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., 2006. ISBN 013168728X.

[48] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Society Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[49] J. Romero, H. Kjellstrom, and D. Kragic. Monocular real–time 3D articulated hand pose estimation. In *Int'l Conf. on Humanoid Robots*, pages 87–92, 2009.

[50] R. H. Liang and M. Ouhyoung. A real–time continuous gesture recognition system for sign language. In *3d IEEE Int'l Conf. on Automatic Face and Gesture Recognition*, pages 558–567, 1998.

[51] H. Kang, C. W. Lee, and K. Jung. Recognition-based gesture spotting in video games. *Pattern Recognition Letters*, 25(15):1701 – 1714, 2004.

[52] K. Kahol, P. Tripathi, and S. Panchanathan. Automated gesture segmentation from dance sequences. In *6th IEEE Int'l Conf. on Automatic Face and Gesture Recognition*, pages 883–888, 2004.

[53] H. Jiang, B. S. Duerstock, and J. P. Wachs. A machine vision–based gestural interface for people with upper extremity physical impairments. *IEEE Trans. on Systems, Man, and Cybernetics: Systems*, 44(5):630–641, 2014.

[54] J. Wan, V. Athitsos, P. Jangyodsuk, H.J. Escalante, Q. Ruan, and I. Guyon. CSMMI: Class–Specific Maximization of Mutual Information for action and gesture recognition. *IEEE Trans. on Image Processing*, 23(7):3152–3165, 2014.

[55] J. S. Wang and F. C. Chuang. An accelerometer-based digital pen with a trajectory recognition algorithm for handwritten digit and gesture recognition. *IEEE Trans. on Industrial Electronics*, 59(7):2998–3007, 2012.

[56] H. K. Lee and J. H. Kim. An HMM-based threshold model approach for gesture recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, 21 (10):961–973, 1999.

[57] M. Elmezain, A. Al-Hamadi, and B. Michaelis. Hand trajectory–based gesture spotting and recognition using HMM. In *16th IEEE Int'l Conf. on Image Processing (ICIP)*, pages 3577–3580, 2009.

[58] N. C. Krishnan, P. Lade, and S. Panchanathan. Activity gesture spotting using a threshold model based on adaptive boosting. In *IEEE Int'l Conf. on Multimedia and Expo (ICME)*, pages 155–160, 2010.

[59] S. Escalera, A. Escudero, P. Radeva, and J. Vitria. Adaptive dynamic space time warping for real time sign language recognition. In *4th CVC Workshop on Research and Development: New Trends and Challenges in Computer Vision*, pages 155–160, 2009.

[60] J. Tian, C. Qu, W. Xu, and S. Wang. KinWrite: Handwriting-based authentication using kinect. In *Proc. 20th Annual Network & Distributed System Security Symposium (NDSS)*, 2013.

[61] J. Guerra-Casanova, C.S. Avila, G. Bailador, and A. de Santos-Sierra. Time series distances measures to analyze in–air signatures to authenticate users on mobile phones. In *IEEE Int'l Carnahan Conf. on Security Technology (ICCST)*, pages 1–7, 2011.

[62] J. B. Kruskall and M. Liberman. The symmetric time warping algorithm: From continuous to discrete. *Time Warps*, pages 125–162, 1983.

[63] R. Oka. Spotting method for classification of real world data. *The Computer Journal*, 41(8):559–565, 1998.

[64] D. S. Hirschberg. Algorithms for the Longest Common Subsequence problem. *J. ACM*, 24(4), 1977.

[65] G. A. Borg. Psychophysical bases of perceived exertion. *Medicine & Science in Sports & Exercise*, 14(5):377–381, 1982.

[66] M. J. Black and A. D. Jepson. A probabilistic framework for matching temporal trajectories: Condensation–based recognition of gestures and expressions. In *European Conf. on Computer Vision (ECCV)*, pages 909–924, 1998.

[67] R. D. Vatavu. The effect of sampling rate on the performance of template-based gesture recognizers. pages 271–278, 2011.

[68] K. Liu, C. Chen, R. Jafari, and N. Kehtarnavaz. Fusion of inertial and depth sensor data for robust hand gesture recognition. *IEEE Sensors Journal*, 14(6): 1898–1903, 2014.

[69] S. Lian, Wei Hu, and Kai Wang. Automatic user state recognition for hand gesture based low-cost television control system. *IEEE Trans. on Consumer Electronics*, 60(1):107–115, 2014.

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

[71] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[72] A. Kurakin, Z. Zhang, and Z. Liu. A real time system for dynamic hand gesture recognition with a depth sensor. In *20th European Signal Processing Conf. (EUSIPCO)*, pages 1975–1979, 2012.

[73] R. P. Mihail, N. Jacobs, and J. Goldsmith. Static hand gesture recognition with 2 kinect sensors. In *Intn'l Conf. on Image Processing, Computer Vision and Pattern Recognition*, volume 2, pages 911–917, 2012.

[74] P. Suryanarayan, A Subramanian, and D. Mandalapu. Dynamic hand pose recognition using depth data. In *20th Int'l Conf. on Pattern Recognition (ICPR)*, pages 3105–3108, 2010.

[75] 3DV systems ltd. link: http://www.3dvsystems.com/.

[76] C. Keskin, F. Kirac, Y.E. Kara, and L. Akarun. Real time hand pose estimation using depth sensors. In *IEEE Intn'l Conf. on Computer Vision Workshops (ICCV Workshops)*, pages 1228–1234, 2011.

[77] L. Billiet, J. M. Oramas, M. Hoffmann, W. Meert, and L. Antanas. Rule-based hand posture recognition using qualitative finger configurations acquired with the Kinect. In *2nd Intn'l Conf. on Pattern Recognition Applications and Methods*, pages 1–4, 2013.

[78] A. Dapogny, R. D. Charette, S. Manitsaris, F. Moutarde, and A. Glushkova. A sign-component-based framework for chinese sign language recognition using accelerometer and sEMG data. hal–00875721, 2013.

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

[80] Y. Yao and Y. Fu. Contour model-based hand-gesture recognition using the Kinect sensor. *IEEE Trans. on Circuits and Systems for Video Technology*, 24(11):1935–1944, 2014.

[81] 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 *5th Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, volume 2, pages 659–663, 1977.

[82] M. Kaâniche and F. Brémond. Recognizing gestures by learning local motion signatures of HOG descriptors. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 34(11):2247–2258, 2012.

[83] M. Kaâniche and F. Brémond. Tracking HOG descriptors for gesture recognition. In *Int'l Conf. on Advanced Video and Signal Based Surveillance*, pages 140–145, 2009.

[84] Y. Yang, I. Saleemi, and M. Shah. Discovering motion primitives for unsupervised grouping and one-shot learning of human actions, gestures, and expressions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 35(7):1635–1648, 2013.

[85] I. Laptev. On space–time interest points. *Int'l Journal of Computer Vision*, 64 (2-3):107–123, 2005.

[86] J. Wan, Q. Ruan, W. Li, and S. Deng. One–shot learning gesture recognition from RGB–D data using bag of features. *The Journal of Machine Learning Research*, 14(1):2549–2582, 2013.

[87] L. Xia, C. C. Chen, and J.K. Aggarwal. View invariant human action recognition using histograms of 3D joints. In *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 20–27, 2012.

[88] S. Fothergill, H. Mentis, P. Kohli, and S. Nowozin. Instructing people for training gestural interactive systems. In *ACM Conf. on Human Factors in Computing Systems*, pages 1737–1746, 2012.

[89] M. E. Hussein, M. Torki, M. A. Gowayyed, and M. El-Saban. Human action recognition using a temporal hierarchy of covariance descriptors on 3D joint locations. In *Int'l Joint Conf. on Artificial Intelligence*, pages 2466–2472, 2013.

[90] O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. In *European Conf. on Computer Vision (ECCV)*, pages 589–600, 2006.

[91] X. Zhao, X. Li, C. Pang, X. Zhu, and Q. Z. Sheng. Online human gesture recognition from motion data streams. In *21st ACM Int'l Conf. on Multimedia (MM '13)*, pages 23–32, 2013.

[92] A. A. Chaaraoui, J. R. Padilla-Lopez, P. Climent-Perez, and F. Florez-Revuelta. Evolutionary joint selection to improve human action recognition with rgb-d devices. *Expert Systems with Applications*, 41(3):786 – 794, 2014.

[93] F. Negin, F. Ozdemir, C. Akgul, K. A. Yuksel, and A. Ercil. A decision forest based feature selection framework for action recognition from RGB-Depth cameras. 7950: 648–657, 2013.

[94] C. Ellis, S. Z. Masood, M. F. Tappen, J.J.Jr. LaViola, and R. Sukthankar. Exploring the trade-off between accuracy and observational latency in action recognition. *Int'l Journal of Computer Vision*, 101(3):420–436, 2013.

[95] S. Cheema, M. Hoffman, and J. J. LaViola. 3d gesture classification with linear acceleration and angular velocity sensing devices for video games. *Entertainment Computing*, 4(1):11–24, 2013.

[96] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I:511–I:518, 2001.

[97] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Trans. on Systems, Man and Cybernetics*, 9(1):62–66, 1979.

[98] L.G. Khachiyan. Rounding of polytopes in the real number model of computation. *Journal Mathematics of Operations Research*, 21(2):307–320, 1996.

[99] A. Li. Approximate lowner ellipsoid, 2008. link: http://www.mathworks.com/matlabcentral/fileexchange/21930.

[100] M. J. Jones and J. M. Rehg. Statistical color models with application to skin detection. 46(1):81–96, 2002.

[101] C. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, 1988.

[102] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Imaging Understanding Workshop*, pages 121–130, 1981.

[103] T. Birdal. Maximum inscribed circle using voronoi diagram, 2011. link: http://www.mathworks.com/matlabcentral/fileexchange/32543-maximum-inscribed-circle-using-voronoi-diagram.

[104] Z. Ren, J. Yuan, C. Li, and W. Liu. Minimum near-convex decomposition for robust shape representation. In *IEEE Int'l Conf. on Computer Vision (ICCV)*, pages 303–310, 2011.

[105] S. Poularakis, A. Briassouli, and I. Kompatsiaris. Full action instances for motion analysis. In *10th Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, pages 37–40, 2009.

[106] I. Katsavounidis, C.-C.J. Kuo, and Zhen Zhang. Fast tree-structured nearest neighbor encoding for vector quantization. *IEEE Trans. on Image Processing*, 5 (2):398–404, 1996.

[107] E. Keogh. Exact indexing of Dynamic Time Warping. In *28th Int'l Conf. on Very Large Data Bases*, pages 406–417, 2002.

[108] J. Alon. *Spatiotemporal gesture segmentation.* PhD thesis, 2006.

[109] S. Lloyd. Least squares quantization in pcm. *IEEE Trans. on Information Theory*, 28(2):129–137, 1982.

[110] L. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286, 1989.

[111] University of Cambridge. Hidden Markov Model Toolkit (HTK). link: http://htk.eng.cam.ac.uk/.

[112] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 23(1):67–72, 1975.

[113] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 26 (1):43–49, 1978.

[114] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, pages 164–171, 1970.

[115] A. Stefan, V. Athitsos, J. Alon, and S. Sclaroff. Translation and scale–invariant gesture recognition in complex scenes. In *Int'l Conf. on PErvasive Technologies Related to Assistive Environments*, page 7. ACM, 2008.

[116] D. Geebelen, J. A. K Suykens, and J. Vandewalle. Reducing the number of support vectors of SVM classifiers using the smoothed separable case approximation. *IEEE Trans. on Neural Networks and Learning Systems*, 23(4):682–688, April 2012.

[117] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Trans. on Neural Networks*, 13(2):415–425, 2002.

[118] T.-F. Wu, C.-J. Lin, and R.C. Weng. Probability estimates for multi–class classification by pairwise coupling. *The Journal of Machine Learning Research*, 5: 975–1005, 2004.

[119] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[120] Ø. D. Trier, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition-a survey. *Pattern recognition*, 29(4):641–662, 1996.

[121] J. McNames. *Innovations in local modeling for time series prediction*. PhD thesis, Stanford University, 1999.

[122] B. Schölkopf, J.C. Platt, J. Shawe-Taylor, A.J. Smola, and R.C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13 (7):1443–1471, 2001.

[123] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[124] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[125] H. K. Lee and J. H. Kim. Gesture spotting from continuous hand motion. *Pattern Recognition Letters*, 19(5–6):513–520, 1998.

[126] Intel. Intel 64 and IA-32 Architectures Software Developers Manual, September 2013. http://www.intel.com/.

[127] D. Y. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham. Fast search algorithms for vector quantization and pattern matching. In *IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, volume 9, pages 372–375, 1984.

[128] Z. Ren, J. Yuan, J. Meng, and Z. Zhang. Robust part–based hand gesture recognition using Kinect sensor. *IEEE Trans. on Multimedia*, 15(5):1110–1120, Aug 2013.

[129] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[130] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[131] S.A. Nene and S.K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 19(9):989–1003, 1997.

[132] A. Guttman. *R–trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[133] J. McNames. A fast Nearest–Neighbor algorithm based on a principal axis search tree. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 23(9): 964–976, 2001.

[134] M. Muja and D.G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 36(11): 2227–2240, 2014.

[135] A. Andoni and P. Indyk. Near–optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th Annual IEEE Symposium on Foundations of Computer Science*, pages 459–468, 2006.

[136] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *IEEE Int'l Conf. on Computer Vision (ICCV)*, pages 750–757, 2003.

[137] B. Kulis and K. Grauman. Kernelized locality–sensitive hashing for scalable image search. In *IEEE Int'l Conf. on Computer Vision (ICCV)*, pages 2130–2137, 2009.

[138] J. He, W. Liu, and S. F. Chang. Scalable similarity search with optimized kernel hashing. In *16th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 1129–1138, 2010.

[139] T.B. Sebastian and B.B. Kimia. Metric-based shape retrieval in large databases. In *16th Int'l Conf. on Pattern Recognition*, volume 3, pages 291–296, 2002.