2006-09-18

Spatiotemporal Gesture Segmentation

Alon, Jonathan

Boston University Computer Science Department

http://hdl.handle.net/2144/1884

Boston University
SPATIOTEMPORAL GESTURE SEGMENTATION

JONATHAN ALON

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

BOSTON UNIVERSITY
BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

SPATIOTEMPORAL GESTURE SEGMENTATION

by

JONATHAN ALON
B.Sc., Tel Aviv University, 1994
M.A., Boston University, 2001

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
2006
Approved by

First Reader

Stan Sclaroff, PhD
Associate Professor of Computer Science
Boston University

Second Reader

Margrit Betke, PhD
Associate Professor of Computer Science
Boston University

Third Reader

Trevor Darrell, PhD
Associate Professor of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Acknowledgments

Embarking on a PhD project is a remarkable undertaking. This journey was made possible because of the help and guidance of various people. Many individuals have provided advice, encouragement, and support to me in my progress toward the completion of this journey, and it is my pleasure to have the opportunity to express my gratitude to all of them.

First and foremost I would like to express my deepest gratitude to my advisor, Stan Sclaroff, for his guidance and support over the past years, and for teaching me all that I know about how to conduct scientific research and how to present my work. I could only hope that a fraction of Stan’s work ethics and high standards would rub off on me.

I would also like to thank my committee members, Margrit Betke, Trevor Darrell, and George Kollios for their critical suggestions and advice, which helped me focus my research and advance my dissertation. The discussions that we had helped me a lot in clarifying certain aspects of the research problem, and in identifying areas for improvement.

I owe my sincere gratitude to Carol Neidle and Robert Lee for always being willing to provide information about sign language, and for their tremendous help with data collection and annotation.

Thanks to all graduate students of the image and video computing group, including Rui, Tian, Murat, Vitaly, Jingbin, Bill, John, Quan, Ashwin, and Jared for creating a fun and vibrant research environment, for having constructive feedback on my work, and for lifting my spirits especially during the last stretch of this journey. Special thanks to Vassilis Athitsos for being a great friend and colleague. I learned a great deal from our collaboration, and from the countless discussions we had on research and life in general. I will surely miss those discussions.

I would like to thank my parents and sister for their love and support from day one. Without a doubt, I wouldn’t be here today if it weren’t for their belief that I can undertake this challenging task.

Finally, I would like to thank my wonderful wife Anat, and my precious kids Maya and
Ori, who I love so dearly, for accepting the long hours spent in the lab, and for making home the best place to come back to.
SPATIOTEMPORAL GESTURE SEGMENTATION

(Order No. )

JONATHAN ALON

Boston University, Graduate School of Arts and Sciences, 2006

Major Professor: Stan Sclaroff, Associate Professor of Computer Science

ABSTRACT

Spotting patterns of interest in an input signal is a very useful task in many different fields including medicine, bioinformatics, economics, speech recognition and computer vision. Example instances of this problem include spotting an object of interest in an image (e.g., a tumor), a pattern of interest in a time-varying signal (e.g., audio analysis), or an object of interest moving in a specific way (e.g., a human’s body gesture). Traditional spotting methods, which are based on Dynamic Time Warping or hidden Markov models, use some variant of dynamic programming to register the pattern and the input while accounting for temporal variation between them. At the same time, those methods often suffer from several shortcomings: they may give meaningless solutions when input observations are unreliable or ambiguous, they require a high complexity search across the whole input signal, and they may give incorrect solutions if some patterns appear as smaller parts within other patterns. In this thesis, we develop a framework that addresses these three problems, and evaluate the framework’s performance in spotting and recognizing hand gestures in video.

The first contribution is a spatiotemporal matching algorithm that extends the dynamic programming formulation to accommodate multiple candidate hand detections in every video frame. The algorithm finds the best alignment between the gesture model and the input, and simultaneously locates the best candidate hand detection in every frame. This
allows for a gesture to be recognized even when the hand location is highly ambiguous.

The second contribution is a pruning method that uses model-specific classifiers to reject dynamic programming hypotheses with a poor match between the input and model. Pruning improves the efficiency of the spatiotemporal matching algorithm, and in some cases may improve the recognition accuracy. The pruning classifiers are learned from training data, and cross-validation is used to reduce the chance of overpruning.

The third contribution is a subgesture reasoning process that models the fact that some gesture models can falsely match parts of other, longer gestures. By integrating subgesture reasoning the spotting algorithm can avoid the premature detection of a subgesture when the longer gesture is actually being performed. Subgesture relations between pairs of gestures are automatically learned from training data.

The performance of the approach is evaluated on two challenging video datasets: hand-signed digits gestured by users wearing short sleeved shirts, in front of a cluttered background, and American Sign Language (ASL) utterances gestured by ASL native signers. The experiments demonstrate that the proposed method is more accurate and efficient than competing approaches. The proposed approach can be generally applied to alignment or search problems with multiple input observations, that use dynamic programming to find a solution.
## Contents

1 Introduction
   1.1 Motivating Applications ........................................ 2
   1.2 The General Problem ........................................... 4
   1.3 Problem Statement ............................................. 7
   1.4 Spatiotemporal Gesture Segmentation ......................... 7
   1.5 Contributions ............................................... 9
   1.6 System Overview .............................................. 12
   1.7 Plan of The Thesis .......................................... 15

2 Related Work
   2.1 Pattern Matching .............................................. 17
   2.2 Efficient Search Methods ................................... 21
   2.3 Visual Detection and Tracking of Hands ...................... 22
   2.4 Data Association ............................................ 25
   2.5 Vision-Based Gesture Spotting and Recognition .......... 27
   2.6 Applications ................................................. 30
   2.7 Summary .................................................. 34

3 Spatiotemporal Matching
   3.1 Dynamic Programming ......................................... 38
   3.2 Dynamic Time Warping (DTW) ................................ 39
   3.3 Continuous Dynamic Programming (CDP) .................... 43
   3.4 Dynamic Space Time Warping (DSTW) ....................... 44
   3.5 Translation Invariance in DSTW ............................ 47
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6 Combining DSTW and CDP</td>
<td>48</td>
</tr>
<tr>
<td>3.7 Matching with Multiple Input Sources</td>
<td>50</td>
</tr>
<tr>
<td>3.8 Complexity</td>
<td>52</td>
</tr>
<tr>
<td>4 Pruning</td>
<td>54</td>
</tr>
<tr>
<td>4.1 Pruning Classifiers</td>
<td>56</td>
</tr>
<tr>
<td>4.2 Spatiotemporal Matching with Pruning</td>
<td>58</td>
</tr>
<tr>
<td>4.3 Learning the Pruning Classifiers</td>
<td>62</td>
</tr>
<tr>
<td>5 Spotting</td>
<td>67</td>
</tr>
<tr>
<td>5.1 Gesture Spotting Algorithm</td>
<td>67</td>
</tr>
<tr>
<td>5.2 Learning Subgesture Relations</td>
<td>71</td>
</tr>
<tr>
<td>5.3 Gesture Verification</td>
<td>72</td>
</tr>
<tr>
<td>5.4 Classical Hypothesis Testing</td>
<td>74</td>
</tr>
<tr>
<td>5.5 Verification Classifier</td>
<td>75</td>
</tr>
<tr>
<td>6 Experiments</td>
<td>80</td>
</tr>
<tr>
<td>6.1 Performance Measures</td>
<td>81</td>
</tr>
<tr>
<td>6.2 Implementation Details</td>
<td>84</td>
</tr>
<tr>
<td>6.3 Digit Recognition Experiments</td>
<td>90</td>
</tr>
<tr>
<td>6.4 ASL Retrieval Experiments</td>
<td>102</td>
</tr>
<tr>
<td>7 Discussion and Future Work</td>
<td>111</td>
</tr>
<tr>
<td>7.1 Contributions of the Thesis</td>
<td>111</td>
</tr>
<tr>
<td>7.2 Strengths and Limitations</td>
<td>116</td>
</tr>
<tr>
<td>7.3 Potential Applications</td>
<td>122</td>
</tr>
<tr>
<td>7.4 Future Work</td>
<td>123</td>
</tr>
<tr>
<td>8 Conclusion</td>
<td>127</td>
</tr>
<tr>
<td>References</td>
<td>129</td>
</tr>
<tr>
<td>Curriculum Vitae</td>
<td>142</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Table of Symbols .................................................. 38

6.1 Classification Accuracy Results ................................. 93
6.2 Subgesture Table ....................................................... 96
6.3 Gesture Spotting and Recognition Results ...................... 97
6.4 Learned Subgesture Table ......................................... 98
6.5 Classification Accuracy Results for Continuous Digit Recognition 100
6.6 List of Signs ............................................................. 107
6.7 Results of Spotting One-Handed Signs ........................ 107
6.8 Results of Spotting Two-Handed Signs .......................... 108
List of Figures

1.1 Illustrative Example ...................................... 3
1.2 Possible Alignments Between a Pair of Patterns ............... 6
1.3 Pattern Finding ........................................... 6
1.4 Hand Detection ............................................ 9
1.5 Pruning of Dynamic Programming Hypotheses ................... 11
1.6 Example False Detection of a Subgesture ...................... 13
1.7 System Flowchart ........................................... 14

3.1 The Dynamic Time Warping Algorithm .......................... 42
3.2 Example Neighborhood in a 3D Dynamic Programming Table .... 46

4.1 Continuous Dynamic Programming with Pruning ................. 60

6.1 Hand Detection Algorithm .................................... 86
6.2 Example Images of the ASL “Y” Hand Shape ......................... 89
6.3 Palm’s Graffiti Digits ........................................ 91
6.4 Example Model Digits ......................................... 91
6.5 Example Correct Match ....................................... 92
6.6 Example False Match ......................................... 93
6.7 Classification Error as a Function of the Number of Candidates .... 94
6.8 Classification Error as a Function of Translation ................. 95
6.9 Digit Models .................................................. 99
6.10 Speedup as a Function of in between Gesture Time ............. 101
6.11 Histogram of Cosine of Angle Difference between Actual and Estimated Direction ......................... 101
6-12 Histogram of 2D Position Displacements .................................................. 101
6-13 ASL Sign Retrieval ....................................................................................... 103
6-14 Classification of Signs based on Number of Hands [11] .............................. 106
6-15 ASL Signs I .................................................................................................. 109
6-16 ASL Signs II .................................................................................................. 110

7-1 Sources of Evidence in Estimating Scale ......................................................... 119
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>DTW</td>
<td>Dynamic Time Warping</td>
</tr>
<tr>
<td>DSTW</td>
<td>Dynamic Space Time Warping</td>
</tr>
<tr>
<td>CDP</td>
<td>Continuous Dynamic Programming</td>
</tr>
<tr>
<td>CDPP</td>
<td>Continuous Dynamic Programming with Pruning</td>
</tr>
<tr>
<td>CDPPS</td>
<td>Continuous Dynamic Programming with Pruning and Subgesture Reasoning</td>
</tr>
<tr>
<td>ASL</td>
<td>American Sign Language</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue</td>
</tr>
<tr>
<td>PDAF</td>
<td>Probabilistic Data Association Filter</td>
</tr>
<tr>
<td>JPDAF</td>
<td>Joint Probabilistic Data Association Filter</td>
</tr>
<tr>
<td>POS</td>
<td>Part of Speech</td>
</tr>
<tr>
<td>LVCSR</td>
<td>Large Vocabulary Continuous Speech Recognition</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Video data is becoming increasingly more available. This can be attributed to a number of recent trends, including the increasing number of video capturing devices, the increase in high capacity storage devices, the increase in availability of fast internet access, and the dropping prices of all of the above. As video databases grow in size so does the need for automated tools for their analysis. One such desirable tool is automatic search and retrieval of particular content of interest from the video database.

The problem of content-based search in video databases is a special case of the more general problem of pattern search in general databases. Automatic search in databases has many important applications in many domains with different types of input including text, DNA sequences, time series (e.g., stock prices), and speech. There are a number of reasons that make search in video databases particularly challenging. First of all, compared to other types of data, the video data size is relatively big, and typically requires larger storage and more processing time. Second, in other types of databases the patterns tend to be easier to extract. For example, in text and bioinformatics databases the patterns are simply strings of characters, and in financial databases asset prices can be recorded directly. In contrast, in video databases, the patterns are not directly observable and they have to be extracted from the images themselves.

In this thesis we focus on the problem of finding motion patterns in video. Methods for finding motion patterns in video have many practical applications including human computer interaction, sign language analysis, automatic video annotation and indexing, and video surveillance, to name a few.
1.1 Motivating Applications

Vision-Based Human Computer Interaction

Human computer interaction (HCI) is one area where the methods proposed in this thesis can be applied. Most people use traditional interfaces such as keyboard or mouse to interact with computers. However, as speech, computer vision, and pattern recognition technologies become more mature, they allow people to use microphones, cameras, and pointer input devices [17] in new ways in order to interact with computers. For example, camera-based systems can give computers the capability of understanding and responding to certain types of human body gestures, such as hand gestures [85]. Compared to many existing interfaces, hand gestures have the advantages of being easy to use, natural, and intuitive. Successful applications of hand gesture recognition systems include computer game control [40], human-robot interaction [109], and sign language recognition [101].

For example, suppose that we want a system that can be an alternative to a standard keyboard for entering text. The user of the system instead of typing on the keyboard, gestures with his hand the characters in front of a video camera. The system takes as input the video stream captured by the camera, and whenever the user completes signing the character the system outputs the recognized character, and uses that either for simple text entry or for controlling another computer application. Figure 1·1 illustrates this idea. The system has a vocabulary of 10 digit gestures, and given the input video the goal of the system is to detect when the user has completed signing a digit and recognize it correctly.

In contrast to a pen-based device, implementing this camera-based system is more challenging due to several reasons. First of all, the pen-based device is a direct-measure device. The tablet’s pressure sensors record the pen trajectory (the sequence of 2D positions) directly. In contrast, the camera-based device has to compute the hand trajectory from image observations. Second, the temporal extent of the character is easy to infer. Whenever the pen starts touching the tablet the character gesture starts and whenever the pen is lifted the character gesture ends (assuming single stroke characters). In contrast,
Given a vocabulary of gestures:

Locate the start and end frame of a gesture within a long video stream, and recognize the gesture.

![Images of gestures](image)

*Figure 1-1: Illustrative example of a digit recognition system.*

in the camera-based device the temporal extent of the character is hard to infer because the hand is always in the camera’s field of view, and there is no clear indication when the gesture starts and ends. The problem of temporal segmentation in the camera-based device leads to a third problem, where some gestures may appear as parts of longer gestures. For example the digit “1” is similar to parts of the digits “7”, “4”, and “9”. Therefore, without addressing this problem the system in certain cases may falsely recognize part of the digit “9” as “1”.

In addition to those difficulties, the usability of such camera-based systems greatly depends on their ability to function reliably in common real-world environments, without requiring the user to wear special clothes or cumbersome devices such as colored markers or gloves [109].

**Retrieval of Signs from Sign Language Video**

Another area where the methods proposed in this thesis can be applied is in the analysis of American Sign Language (ASL) video. ASL is a gestural language, with a different structure than spoken or written English, and is used primarily by the deaf community in
North America. The proposed methods would enable a broad range of applications, but here we use two examples to illustrate. The first application is motivated by the fact that most sign language dictionaries are ordered lexicographically by the corresponding English word. This makes it difficult for novice students to look up an unfamiliar sign: a common way of learning sign language is by watching example videos of sign language narrative. If the student sees in the video an unfamiliar sign that he wants to learn, a search in a lexicographical order would not be of much use. In the proposed application, which we call The Visual Sign Language Dictionary, the user would sign the unfamiliar sign in front of a camera, and then submit the sign video clip as a query to the system. The system will then retrieve the most similar sign examples from the dictionary.

The second application is motivated by the growing need for automated tools for sign language video annotation. Currently, all annotations are performed by linguists with special-purpose software [78]. In the proposed application, which we call Sign Spotting, the input to the system consists of a long video clip (e.g., a sign language story), and given a query depicting a sign, the goal is to retrieve all the occurrences of that sign in the video clip, and to estimate each sign’s temporal extent. Solving this problem will be an important step towards a fully automatic annotation system for sign language video.

1.2 The General Problem

The general problem underlying the example systems described in the previous section, and which is addressed in this thesis, is that of finding pattern(s) in an input stream. Instances of this general problem have appeared in several fields under different names, including string matching in computer science and bioinformatics [47], subsequence matching in data mining [39], word spotting in speech recognition [94], and gesture spotting in computer vision [67]. This section describes some of the challenges that are common to most of those problems.

Methods for finding occurrences of a pattern in an input signal must match the pattern with subsequences of the input. Matching is implemented by mathematically defining a
distance measure between a pair of patterns. There is no single distance measure that is suitable for all applications, and the success of an application greatly depends on the choice of a suitable distance measure. Before defining a distance measure between a pair of patterns, it is often required to define a distance measure between a pair of pattern elements. For example, before matching a pair of strings, it is required to define a distance measure between a pair of characters. A reasonable measure would assign a distance of zero if the characters are equal, and one otherwise. The distance between a pair of strings can then be defined by summing all the pairwise distances between characters. Using this particular distance, two strings would match exactly if the distance between them is zero. A distance value other than zero would indicate how different the strings are.

In some applications computing a distance between a pair of patterns may require finding a correspondence (or alignment) between their elements. Figure 1-2 illustrates this idea. In the figure there are two similar patterns of the digit “3” with six elements each. The figure on the left depicts a one-to-one correspondence, where every element of one pattern is matched with exactly one element of the other pattern. The figure on the right depicts a more general correspondence, where every element of one pattern can be matched with multiple elements of the other pattern. The latter correspondence seems more reasonable for this particular matching problem. At the same time, finding the correspondence with the smallest distance becomes computationally more expensive, because the number of possible correspondences is exponentially high.

The problem of finding the distance between a pair of patterns becomes even more challenging when one of the patterns is embedded in a long input stream (see Figure 1-3). In addition to finding a meaningful correspondence between the patterns elements, the start and end points of the embedded pattern must be estimated. A brute force method for finding a pattern in an input stream is to slide the pattern over the input, and compute the distance between the pattern and all input subsequences formed by all possible start and end points. A pattern is found in the input whenever a corresponding subsequence has a sufficiently small matching cost. The brute force method leads to a very expensive
Figure 1-2: Possible matching alignments between a pair of patterns of the digit “3”. (a) One-to-one matching, where each element of one pattern matches exactly one element of the other pattern. (b) A more general matching, where a single element of one pattern can match multiple elements of the other pattern. On the left of each figure, pairs of elements are matched with a line. On the right of each figure, the alignment is represented by the shaded cells in a table, where each table cell corresponds to a possible match between a pair of individual elements.

Figure 1-3: Finding a pattern within a long sequence. The pattern in the boxed figure is an example of a hand signed “5” digit gestured in front of a video camera. The x coordinate of the hand trajectory extracted from the video is plotted over time. The input contains another example of the digit “5” indicated by the dashed lines. The goal of pattern spotting is to detect the pattern within the input, and estimate the corresponding start and end points.

search. Furthermore, the matching procedure may be carried out for multiple patterns of interest, and therefore the search time may further increase in proportion to the number of patterns.
1.3 Problem Statement

The general problem addressed in this thesis is finding pattern(s) in an input stream. To be concrete we will define the gesture spotting problem, keeping in mind that other instances of the general problem can be cast in a similar way using slightly different terminology. Given $G$ gesture models $M^g$ ($1 \leq g \leq G$), where $g$ is a gesture class label, and an input stream of features $Q = Q_1, Q_2, \ldots, Q_j, \ldots$, we want to find the start and end frames $(j_s, j_e)$ of every gesture appearing in the input, and recognize it as one of the gesture classes $g$. This problem definition includes as a special case the problem of spotting a single gesture in the input stream ($G = 1$). In the vision-based gesture spotting problem considered in this thesis the input is a video clip or an image sequence, and each input feature vector $Q_j$ is computed from the video frame $j$.

The second problem we will define is the isolated gesture recognition problem. Given $G$ gesture models $M^g$ ($1 \leq g \leq G$), and an isolated gesture query $Q = Q_1, Q_2, \ldots, Q_n$, we want to find the best matching model (with respect to a given distance measure) or alternatively recognize the query as one of the gesture classes $g$.

1.4 Spatiotemporal Gesture Segmentation

In order to recognize hand gestures in a continuous video stream, an end-to-end vision-based system needs to perform spatial and temporal gesture segmentation. Spatial segmentation means determining where the gesturing hand is located at each frame, as opposed to determining the spatial extent of the gesture.) Temporal segmentation means determining when the gesture starts and ends. In most vision-based gesture recognition systems, information flows in a bottom-up way: the high-level recognition process assumes that spatiotemporal segmentation has already been performed as a preprocessing step. However, in more natural settings, the gestures of interest are embedded in a continuous stream of motion, and may occur in front of a cluttered, non-static background. In such settings, both spatial and temporal segmentation of gestures are challenging tasks to perform reliably in a purely
bottom-up fashion. If a gesture recognition system relies on the availability of spatiotemporal gesture segmentation “black boxes,” then the range of settings in which the system can be applied is severely limited. For example, it is noted in [82] that most vision-based sign language recognition systems assume that the subjects wear long-sleeve shirts and/or that the hands are the fastest moving objects in the scene, in order to extract the hand location as a preprocessing step.

The key novelty of our method is that gesture recognition does not require accurate spatial or temporal segmentation as a preprocessing step, and information flows both bottom-up and top-down, between the segmentation and the recognition processes. In settings where existing vision methods for hand localization and/or temporal gesture segmentation can be reliably performed as preprocessing, such methods can be used within our framework to improve recognition accuracy. At the same time, in settings where such vision-based preprocessing is beyond the current state of the art, and existing gesture recognition algorithms are not applicable, our method can still be used. By integrating information from the low-level segmentation tasks and the high level recognition task, the proposed algorithm simultaneously recognizes gestures and identifies the best spatiotemporal segmentation hypothesis for each gesture.

In the literature, the problems of spatial and temporal gesture segmentation have mostly been addressed separately. The problem of spatial segmentation was recently addressed by the methods proposed in [5, 96], but those methods still assume that the start and end frames of each gesture are either known or can be reliably estimated. The problem of temporal segmentation, also known as gesture spotting, has been addressed in [3, 17, 67, 75, 80, 125, 129], but those methods assume spatial segmentation, i.e., that the hand can be unambiguously localized at each video frame. Our method assumes neither spatial nor temporal segmentation are known. Instead, it tries to accomplish both segmentations simultaneously, and is therefore applicable in a much wider range of gesture recognition settings.
Figure 1.4: Detection of candidate hand regions based on color and motion. Clearly, color and motion are not sufficient to unambiguously detect the gesturing hand since the face and the arms have similar features. Nonetheless, for this particular scene, the gesturing hand is consistently ranked among the top 5 candidate detections.

1.5 Contributions

In order to achieve the objective of spatiotemporal gesture segmentation a number of problems must be addressed. We identify three key problems that limit existing gesture spotting methods, and propose methods for solving those problems. The proposed methods are the main contributions of this thesis.

Contribution 1: Matching with Multiple Observations

The first problem is the assumption, made by most existing bottom-up methods, that the gesturing hand can be reliably located in every frame of the input sequence. Despite recent advances in hand detection [33, 81] and hand tracking [51, 105, 107], in many real life settings this assumption cannot be satisfied. Common challenges that make hand detection extremely difficult are changing illumination, low quality video and motion blur, low-resolution hand images, temporary occlusion, and cluttered background. For example, in Figure 1.4 hand detection based on motion and skin color yields multiple hand candidates, and the top-ranked candidate may be incorrect. Other visual cues commonly used for hand detection such as edges, and background subtraction [24, 72] may also fail to unambiguously locate the hand in the image when the face, non-gesturing hand or other “hand-like” objects are moving in the background.
Instead of assuming perfect hand detection, we make the milder assumption that existing hand detection methods can produce, for every frame of the input sequence, a relatively short list of candidate hand locations, and that one of those locations corresponds to the gesturing hand. In order to accommodate multiple candidate detections in every input frame we propose a novel spatiotemporal matching algorithm. The algorithm aligns subsequences of the input video to a model gesture in time, while at the same time it identifies the most likely hand location out of the multiple candidates available for every input frame. The proposed algorithm can be generally applied to other pattern matching problems where multiple input observations are available at every time instance.

**Contribution 2: Efficient Search using Pruning Classifiers**

The second problem that gesture spotting algorithms face is expensive time complexity. Several methods for spotting gestures in continuous streams [67, 75, 80] are extensions of the dynamic programming (DP) methods used for recognizing isolated gestures. Finding the optimal matching between a gesture model and an input sequence using brute-force search would involve evaluating an exponential number of possible alignments. The key advantage of DP is that it can find the best alignment in time, which is linear in the number of input frames. However, overall recognition time is also linear in the number of models, and the number of model states. This may be insufficient for applications that need to evaluate a large number of hypotheses (due to e.g., large size of gesture vocabulary), and/or applications that may be required to respond to user commands in real-time.

To reduce the search time even more, several pruning methods have been proposed in the speech recognition community [52] and successfully applied in large vocabulary sign language recognition [117]. Those pruning methods typically sacrifice little accuracy for significant speedup. Beam-search [52] is one such method that prunes unlikely alignment hypotheses, and keeps only promising hypotheses with matching scores, which are within a threshold (the beam width) from the matching score of the best hypothesis. However, the beam width is often manually specified in ad hoc way.
Figure 1-5: Pruning: example dynamic programming table for matching an input gesture (horizontal axis) to a model gesture (vertical axis) of the same digit class “6”. Every entry in the table corresponds to a matching between a subsequence of the input and a subsequence of the model. The white-colored cells are pruned by the DP algorithm, the black-colored cells are visited by the DP algorithm, and the red colored cells correspond to the optimal alignment between the model and the input. The pruning in the left figure is conservative but does not prune the optimal path. The pruning in the right figure is so aggressive that it prunes the optimal path. The result of the proposed learning-based pruning method is shown in the middle figure. It is more aggressive than the pruning on the left but it does not prune the optimal path as does the pruning on the right.

In this thesis, we propose a more principled pruning method that does not require manual setting of thresholds. The proposed method treats pruning as a binary classification problem. In light of this view, pruning classifiers can be learned from training data off-line. The proposed classifier learning algorithm aims at maximizing the efficiency, or the amount of pruning, while at the same time minimizing the chance of falsely pruning correct alignment hypotheses. (See Figure 1-5). A cross-validation framework is used to increase the generalization of the pruning classifiers to yet unseen gesture sequences. The proposed learning-based pruning algorithm can be generally applied to speed up any dynamic programming-based search method.

Contribution 3: Reasoning about Nested Patterns

The third problem that has to be addressed by a gesture recognition algorithm is how to determine a good set of spotting rules to detect promising candidate gesture boundaries. Simple spotting rules, such as ruling out unlikely models based on model-specific thresholds
and selecting the model with the highest matching score [67, 75], may be insufficient to
detect the correct candidate models. For example, such spotting rules may fail if one of the
gestures matches well with parts of other gestures. (See Figure 1-6). The problem of nested
patterns can be addressed by combining the score of a spatial classifier with a low-level
temporal event, such as change in gesture velocity, in order to determine gesture completion
[17]. However, such temporal events may be difficult to infer from image sequences.

We propose learning, for each gesture class, the set of “subgesture” classes, i.e., the set
of gesture models that are similar to subgestures of that class. While a gesture is being
performed, it is natural for these subgesture classes to cause false alarms. When such a
situation occurs we test if the longer gesture is active (plausible): if the longer gesture is
not active we can recognize the shorter gesture. If the longer gesture is active then we wait
until either it becomes inactive, in which case we recognize the shorter gesture, or until it
completes, in which case we recognize the longer gesture. Our subgesture reasoning can
thus reliably recognize and avoid the bulk of those false alarms. Nested patterns appear
in other types of input such as text, and therefore similar reasoning can be applied there.

1.6 System Overview

This section introduces an overview of the system proposed for spotting and recognition
of patterns embedded in a long continuous input stream. For illustration purposes we
focus on spotting and recognition of dynamic hand gestures in video sequences. In this
context, the distinguishing characteristic of the proposed framework is that it can segment
a particular hand gesture both in space and time without relying on extremely accurate
hand detection.

The overall framework, depicted in Figure 1-7, consists of three major components:
hand detection and feature extraction, spatiotemporal matching, and gesture spotting and
recognition. The offline learning phase (marked by the dashed arrows) consists of learning
the gesture models, the pruning classifiers, and the subgesture relations between gestures.

In the input sequences used for the offline learning phase the locations of the gesturing
hands are manually labeled. Alternatively, the user is asked to wear colored gloves. This enables the hand detection algorithm to reliably localize the gesturing hand, and the feature extraction algorithm to extract a single feature vector per frame.

In the input sequences used for the online recognition phase the user gestures naturally in front of the camera without any aiding device; therefore, the hand detection algorithm can no longer localize the gesturing hand with certainty, but instead generates a short list of hand candidate windows in every frame. The feature extraction module extracts multiple feature vectors (one feature vector per candidate window). The resulting time series of multiple feature vectors is then matched with the learned models in both space and time. During the matching process the learned pruning classifiers reject unlikely matching
hypotheses. Finally, the matching costs of the different models are fed into the gesture spotting and recognition module, which applies a set of spotting rules and subgesture reasoning to spot and recognizes the gestures performed by the user.

We use the proposed method to implement a digit recognition system that can recognize the user’s gestures in real time. Compared to many existing interfaces, that were successfully applied to control computer robots [109] and computer games [40], our method can function reliably even in the presence of clutter or moving background, without requiring the user to wear special clothes or cumbersome devices such as colored markers or gloves. In addition, we implement a new type of computer vision application: a gesture-finding tool that helps users locate gestures of interest in a large video sequence or in a database of such sequences. In the experiments we use this tool to find occurrences of American Sign Language (ASL) signs in video streams of sign language narrative signed by native
1.7 Plan of The Thesis

This chapter presented the general problem of finding patterns in input streams, and the more specific problem, that is addressed in this thesis, of spotting gestures in video streams. The major challenges of this problem have been described, and a general description of the methods proposed to solve this problem has been provided. The rest of the thesis is organized as follows:

Chapter 2 discusses related problems from several different fields, including pattern matching from computer science, efficient search methods from speech recognition and data mining, visual hand tracking, data association, and gesture recognition from computer vision, and applications of gesture recognition to human computer interaction, sign language analysis, and handwriting recognition. Existing methods are compared and contrasted with the methods proposed in this thesis in the context of the gesture spotting problem.

Chapter 3 describes the proposed spatiotemporal matching algorithm. A number of dynamic programming-based methods are first presented that can match sequences that are either segmented in space or time. After the necessary background is laid out, the chapter describes our new spatiotemporal matching algorithm that requires neither temporal nor spatial segmentation.

Chapter 4 shows how to speed up the spatiotemporal matching algorithm by using pruning. The chapter starts by describing the idea of treating pruning as a binary classification problem. Example pruning classifiers are given, and a method for learning those classifiers in a principled way is also presented.

Chapter 5 describes the spotting method used to detect gestures in an input stream. The set of spotting rules that are used in the method are derived. In particular, the chapter introduces the concept of subgesture relations, and how they are learned (from training data) and used in order to improve the gesture spotting system’s accuracy.
Chapter 6 presents the quantitative experiments conducted with the proposed system, and the performance measures used for its evaluation. A description of the image capturing setup used for data collection, the computer setup used for running the experiments, and other implementation details is given. Finally, the performance of the proposed system is evaluated on two challenging tasks, vision-based digit recognition and ASL sign retrieval.

Chapter 7 provides a summary of the work presented in this thesis, and places the thesis contributions in the context of computer vision and in the larger context of search and pattern matching problems. The strengths and limitations of the proposed system are discussed. The chapter concludes with a list of open problems and possible future research directions.
Chapter 2

Related Work

The framework presented in this thesis is general. It can be used to find patterns in text, speech and video. In this thesis, we chose to apply the framework to spot gestures in video streams. This section presents related works from many different areas: pattern matching mainly from the area of computer science and database search; efficient search methods mainly from the areas of artificial intelligence (AI) and speech recognition; hand detection, tracking and gesture recognition from the area of computer vision; and finally applications in handwriting recognition, human computer interaction, and sign language recognition.

2.1 Pattern Matching

Pattern matching is a problem that has many important applications for different types of input including text, DNA sequences, time series (e.g., stock prices), speech, and video. For example, the string-matching problem arises frequently in text-editing programs, where the goal is to find all occurrences of a particular pattern in a text\cite{27}. Formally, the string-matching problem can be defined as follows\cite{27}. We assume that the text is an array $Q[1..n]$ of length $n$ and that the pattern is an array $M[1..m]$ of length $m$. We further assume that the elements of $M$ and $Q$ are drawn from a finite alphabet $\Sigma$ (e.g., $\Sigma = a, b, \ldots, z$). We say that $M$ occurs with shift $s$ in text $Q$ if $Q[s + 1..s + m] = M[1..m]$. If $M$ occurs with shift $s$ in $Q$, then $s$ is a valid shift. The string-matching problem is the problem of finding all valid shifts with which a given pattern $M$ occurs in a given text $Q$.

The naive brute-force string-matching algorithm has worst-case running time $O(nm)$.
The naive algorithm can be interpreted graphically as sliding a window containing the pattern over the text, and finding for which shifts all of the characters in the window equal the corresponding characters in the text. The naive algorithm is inefficient because information gained about the text for one value of $s$ is totally ignored in considering other values of $s$. Many efficient algorithms, such as Karp-Rabin [57], Knuth-Morris-Pratt [61], and Boyer-Moore [21], have been proposed that make use of information already gained about the text for a previous shift. Those algorithms may still have worst-case running time of $O(nm)$ but they often work much better on average and in practice. If the text to be searched is fixed then preprocessing it can speed up the search even more. A suffix tree [74] can be constructed from a given text in $O(n)$ time [111], and can be used to find a new pattern using only $O(m)$ character comparisons.

The string-matching algorithms described so far are exact matching algorithms. That is, those algorithms assume that the elements of the pattern and text are discrete, and that a valid match corresponds to a pattern and a substring of a text being identical in all elements. More flexible methods have been proposed that allow for approximate matching between pairs of strings. Approximate matching methods allow for a certain set of transformations in the matching, including repetitions, deletions, substitutions, and insertion of individual elements. For computing matching scores those methods can employ either similarity measures (e.g., the Longest Common Subsequence [47]) or distance measures, e.g., Dynamic Time Warping [64], and the Levenshtein distance [68] (aka, edit distance). Those methods can be extended to handle sequences of real-valued elements which often arise in many different applications including speech and computer vision. In addition, the methods can be used for matching entire sequences (whole matching) as well as for finding patterns in longer sequences (subsequence matching).

In order to compute a similarity score, many of those more sophisticated similarity measures have to solve a non-trivial alignment problem to account for the different transformations. For example, Dynamic Time Warping and has been successfully used in gesture recognition [28, 37, 5] due to its ability to model variation in gesture duration or gesture
speed (which is similar to its ability to model repeating characters in matching strings).

The alignment problem is to find which elements of one sequence correspond to which elements of the other sequence. The number of possible alignments is exponential in the sequence length. A brute-force search for the optimal alignment yielding the best matching score is therefore impractical. Fortunately, dynamic programming methods [13] can find the optimal alignment in polynomial time, by dividing the global optimization problem into a sequence of local optimization problems, and then combining the local solutions into the desired global solution.

An alternative to using similarity measures for matching sequences is to use statistical similarity models. The hidden Markov model (HMM) [88] is an example of such a model that is widely used for matching and recognition of sequences. The HMM is a probabilistic finite state machine that has a number of states, each of which have a set of transition links to other states, possibly including a link to the same state, called a “self-transition.” Associated with each transition is a transition probability, and associated with each state is an observation probability. Given a sequence of observations, the Viterbi (dynamic programming-based) algorithm [114] can be used to compute the optimal state sequence (temporal alignment), and the corresponding likelihood score of the sequence given the HMM model.

It has been noticed by other researchers that although the Dynamic Time Warping measure and hidden Markov model are seemingly different, they share common features and in fact DTW can be seen as a special case of HMM [79, 9]. The DTW [64] is a deterministic example-based method that computes a matching cost between a pair of sequences, while accounting for non-linear transformations in time. The HMM [88] is a probabilistic model-based method that computes the likelihood score of the sequence, while accounting for non-linear transformations in time, through the use of self-transitions. Both methods use dynamic programming for computing the matching cost.

One key difference between HMM and DTW is that the HMM classifier can be learned from a set of training examples. Similar to other classifiers, as the number of training
examples increases, and the distribution of the training set becomes more representative of the true distribution, then the prediction accuracy of the classifier on new test sequences also tends to increase. Another difference is that HMM can skip over states moving forwards or backwards. However, in practice since gestures move forward in time, they can be adequately modeled by a subclass of HMM models called left-right HMMs. If, in addition, the probability of staying at the same state is equal to the probability of transiting to the next state, and the HMM model states are identified with their observation density means (assuming a simple Gaussian probability model) then the HMM essentially transforms into a DTW model.

DTW has been a popular method for speech recognition in its early days [52]. It was originally intended to recognize spoken words of small vocabulary [64, 88], when only few examples of each word were available. However, as gesture vocabularies grew in size, and more annotated speech corpora became available (and consequently larger amounts of training data) the HMM became the predominant model. A similar development occurred in the area of gesture recognition: DTW can be used to successfully recognize a small vocabulary of gestures [28, 37], when only a few examples are available for every gesture, while HMM-based approaches can be used to recognize larger vocabularies of gestures [117], when a relatively large number of training examples is available for every gesture.

The gesture spotting problem considered in this thesis can be viewed as a pattern matching problem. In the gesture spotting problem the input stream is a sequence of observations (or features) extracted from the input video. A “gesture pattern” can be a single example sequence of observations extracted from a sequence of images, or a model learned from several examples of the same gesture. The real-valued features often computed from images can be converted to discrete-valued features through a categorization method, such as vector quantization [45]. This enables the use of string matching algorithms. At the same time, the categorization procedure is an approximation, and there are some practical issues involved, such as selecting the correct number of categories, and effectively categorizing high-dimensional data. We therefore choose to represent the gestures and the
input by sequences of multidimensional real-valued observations, or multidimensional time series.

Example gestures from the same class can vary in their trajectory as well as in their duration, and therefore a flexible similarity measure that can model temporal variation has to be employed for matching gestures. In this thesis, dynamic programming is used to compute the alignments between a gesture and subsequences of the input, and the corresponding matching costs. The proposed spatiotemporal matching algorithm and the pruning method can be applied to improve the robustness and efficiency of dynamic programming-based methods, such as DTW, HMM, and their variants.

2.2 Efficient Search Methods

The problem of continuous gesture recognition is to recognize a sequence of gestures from an image sequence. This problem is similar to the problem of continuous speech recognition, where the goal is to recognize a sequence of words from a speech utterance. It is therefore not surprising that many of the approaches used to solve those problems are similar.

In continuous speech recognition, there are two families of methods for searching the most probable word sequence hypothesis given the acoustic input and the language model: the Viterbi search and tree search methods [52]. The Viterbi search finds the most likely sequence of words to have caused the observed acoustic data. Each word is represented by a HMM, and the search is over a composite HMM model or a HMM network. In a tree search method the tree branches are labeled with the various words of the vocabulary. The number of branches leaving each node is equal to the vocabulary size, and the depth of the tree is equal to the number of words in the sequence. The tree search methods seek a path through a tree of possible hypotheses.

Both the Viterbi search and tree search are impractical for large vocabularies of words. A number of efficient methods have been proposed to alleviate their high computational cost and memory requirements. Those methods have also been successfully applied in large vocabulary gesture recognition systems. Those methods include fast matching [43], N-best
pass [117], network pruning and beam search [12, 43, 117]. Language models that have been used to speedup the search include unigram and bigram models [43, 117], as well as a strongly constrained part-of-speech (POS) grammar [102].

In order to scale the methods to large vocabularies and reduce the amount of training data, some researchers defined gesture or sign subunits, similar to phonetic acoustic models in speech. Each gesture model then consists of a concatenation of HMMs which model subunits. The number of subunits (a few tens) is typically smaller than the number of gestures (a few hundreds or thousands), therefore reducing the search space significantly. The subunit models may be learned using unsupervised learning [12] or using linguistic knowledge [115, 117].

In this thesis we propose an efficient method for spotting gestures. The method that is most related to ours is the beam-search [52]. Beam search can be used in dynamic programming to prune unlikely hypotheses from the search, and keep only promising hypotheses with matching scores, which are within a threshold (the beam width) from the matching score of the best hypothesis. The beam width is often set manually in an ad hoc fashion. In this thesis, we propose to view pruning as a binary classification problem. Pruning classifiers are learned offline from training data with the objective of maximizing pruning while minimizing the chance of falsely pruning the optimal alignment between the gesture and the input. The pruning classifiers are learned in a cross-validation framework and are shown to have good generalization properties. This can be seen as a more principled way to learn the beam width. Furthermore, different types of classifiers with different types of features can be used for pruning, as opposed to pruning only based on matching scores.

2.3 Visual Detection and Tracking of Hands

This section describes methods for locating hands in images and tracking them from frame to frame. In a bottom-up approach to gesture recognition hand tracking is used to extract the trajectory of the hand. The trajectory is then classified to one of a pre-specified set of gestures.
Visual detection and tracking of hands is a challenging problem with applications in gesture recognition, sign language recognition, and human computer interfaces [85]. The appearance of a hand in an image can be explained by the relative pose of the hand with respect to the camera, and by the hand shape or posture. The relative pose of the hand can be described by the six degrees of freedom of a general rigid transformation, and its shape can be described by the 20 joint angles of its 15 joints [123]. The high dimensionality of this configuration space results in a large variation in hand appearance. Furthermore, many different hand configurations may approximately match a given image due to the many-to-one nature of projection and due to self-occlusion. The high dimensionality of the hand configuration space, the large variation in hand appearance, and the ambiguities inherent in images of articulated objects like the hand, are the major challenges of visual hand detection and tracking.

Over the years two major categories of hand trackers have emerged: 3D model-based and 2D appearance-based [99]. Methods in the first category attempt to fit the projected 3D model to image observations [92, 123, 103, 107]. Since the 3D hand model is only indirectly related to image observations the estimation of the posterior distribution of model configurations is intractable [107], and recent work has mostly focused on approximating the posterior. Extended [92] and unscented [104] Kalman filters approximate the posterior by a single Gaussian, while particle filters and other nonparametric distributions approximate the posterior distribution by a weighted set of particles.

Another problem that 3D model-based trackers must address is the high dimensionality of the hand configuration space. The curse of dimensionality leads to a sparse posterior with sharply peaked modes. As a result the tracker tends to concentrate in only a few of the most significant modes. The correct mode can be missed and as a result the tracker may drift and eventually fail. Several approaches have been proposed to deal with this problem. One approach to circumvent this problem is by considering a more simplified hand model with a limited range of hand motions [70]. Another approach is to use a stronger dynamic model, potentially learned from training data [123]. An alternative way is to represent
the hand structure by using a graphical model [107] or a Bayesian network [122], and use appropriate statistical inference algorithms to estimate the posteriors [107, 122]. This approach often leads to more accurate but more computationally-intensive trackers.

A second category of methods try to match the possible 2D appearances of the target object to the input image [32, 127]. The methods are fairly robust to self-occlusion since they directly compare intensity patterns between the input image and the registered target. On the downside, only coarse hand shape can be recovered due to the more simple 2D representation. Furthermore, many of those methods are limited to a particular hand view. To constrain the tracking problem many approaches have used fixed or known backgrounds [120], colored gloves [101], limited movement or markers [19]. For example, some approaches use color to find coherent skin regions. Those approaches rely on the fact that human skin color is relatively uniform and can be modelled adequately with a single Gaussian [120]. Although more sophisticated color models can be used [53], color alone generates ambiguity between the hands and other similarly colored objects. This problem can be addressed through the fusion of additional visual cues such as edges and motion [98]. More recently, researchers have adopted the popular approach of boosted classifiers, which was originally applied to detect faces [112] and pedestrians [113], to detect hand shapes in gray-scale images [81, 63].

Despite the recent advances of visual hand tracking most vision-based hand trackers still suffer from two major drawbacks: they need to be manually initialized, and they cannot recover when they lose track. One strategy to overcome the problem of manual initialization and tracking failure is to treat tracking as object detection at each frame. Thus, if the target is lost in one frame, this does not affect any subsequent frame [93, 7]. Tracking can also be combined with detection to improve the robustness of visual trackers [119]. Another strategy to overcome tracking failures is to maintain multiple hypotheses, for instance, as is done in the particle filtering framework. The advantage of particle filters over filters with unimodal Gaussian observation densities is that they can deal with clutter and ambiguous situations effectively, by maintaining multiple hypotheses and not
committing to the single best hypothesis. This allows the tracker to avoid tracking failures that would otherwise occur in ambiguous frames.

In this thesis, we use similar ideas to address the problem of tracking failure, but we attack this problem from a different direction. In the context of hand gestures, the problem we aim to solve is finding occurrences of a particular gesture of interest in an input video. Since the computer can learn a model for the gesture of interest, this model can be used both to provide a matching cost for recognition, and to guide the search for the best hand hypothesis in every frame. To accomplish this objective we propose a spatiotemporal matching algorithm that can accommodate multiple hand hypotheses in every frame. By fusing information from low-level detectors with high-level knowledge about the gesture model, the proposed algorithm can essentially circumvent the problem of tracking. Therefore, no manual initialization is required, and tracking failures that would cause existing gesture recognition systems to break down, would not cause the proposed system to break down. At the same time, assumptions often made by trackers about motion coherence and motion consistency can be incorporated in the algorithm as well. Furthermore, tracking can be incorporated in the algorithm to improve gesture recognition accuracy, but the recognition system would not break down if tracking fails.

2.4 Data Association

In the context of multiple target tracking the data association problem is to determine the correspondence between the current measurements and the tracks of the moving objects [10]. In computer vision this problem is also known as the motion correspondence problem [30]. Solving this problem is often required for estimating camera motion [31], 3D scene geometry, and optical flow, and for visual tracking of objects [31, 69, 90].

Several data association techniques applied to computer vision problems were surveyed in a review by Cox [30]. The simplest suboptimal algorithm is the nearest-neighbor algorithm, that assigns each measurement to the closest corresponding feature. For this algorithm there is always a chance that the association is incorrect, and this can lead
the tracking filter to diverge. The nearest-neighbor algorithm makes assignments based solely on the current frame. Better correspondences can be achieved by postponing the assignment decision in the hope that future measurements in subsequent frames will clarify current ambiguities. This idea is implemented in the track-splitting, joint likelihood tracking, and multiple hypothesis tracking algorithms, which are described next.

The track-splitting filter forms a tree, where each branch denotes an alternative assignment of the measurements. Assignment decisions are postponed until additional measurements have been gathered to either support or refute earlier assignments. Due to the combinatorial nature of track splitting, pruning methods are often required for practical implementation. Those pruning methods include: deleting unlikely tracks, merging similar tracks, and maintaining the $K$ most likely tracks. Merging tracks is necessary because tracks may share measurements, but this is physically unrealistic. It is more reasonable that a measurement originates from a single object or feature. This problem can be solved via partitioning of measurements into disjoint tracks. Alternatively, the partitioning can be made probabilistic using an appropriate observation model [69]. The multiple hypothesis filter [31, 22] extends the methods described so far by handling track initiation and termination. Track initiation is required when new objects or features come into view for the same time. Track termination is required when objects or features are occluded or leave the sensor’s field of view.

The main problem with the methods discussed in the previous paragraph is their combinatorial nature, which demands large computation and memory resources even if significant pruning is applied. Probabilistic data association filters were introduced in order to reduce the computation and memory requirements, and enable real-time applications. Those filters associate all the measurement with all the tracks. They assign to every measurement-to-track pair a weight that represents the probability that the measurement has originated from that track. The original probabilistic data association filter (PDAF) assumes the existence of a single target. The joint probabilistic data association filter (JPDAF) extended this to a fixed known number of targets. JPDAF was used for head tracking to associate
edge measurements with a head contour [25]. A constrained JPDAF was also used for tracking people by modeling the constraints between the different human body parts [90].

The methods described in this thesis can be thought of as data association methods but they differ in one important respect. The data association methods presented in this thesis are used for recognition and not for tracking. While the objective of tracking is to follow the trajectory of an object (or to follow trajectories of multiple objects), the objective of recognition is to classify a trajectory, or more generally a multidimensional time series, to one of a predefined set of categories. Even a sophisticated hand tracking algorithm, which models track initiation and termination, may fail when the gesturing hand is temporally occluded, or when the hand enters or leaves the camera’s field of view. In contrast, the proposed gesture recognition algorithm will not fail under these circumstances. The “worst” that can happen is that none of the gesture models will be recognized, but the system will not break down. Eventually, one of the gesture models will match the input observations, and when it does that model will be recognized. We present two variants of the gesture recognition algorithm: the first variant assumes the existence of a single gesturing hand, similar to the original PDAF. The second variant can handle two-handed gestures, similar to the JPDAF. Although tracking is not explicitly modeled, the gesture track can be recovered by tracing back the optimal path of the dynamic programming table corresponding to the recognized gesture model.

2.5 Vision-Based Gesture Spotting and Recognition

In most dynamic gesture recognition systems (e.g., [28, 35, 37, 102]) information flows bottom-up. Lower-level modules perform spatial and temporal segmentation and extract shape and motion features. Those features are passed into the recognition module, which classifies the gesture [85]. In bottom-up methods, recognition may fail in the absence of reliable spatial and/or temporal segmentation.

The Dynamic Time Warping (DTW) algorithm, for example, which was originally intended to recognize a small vocabulary of spoken words [64, 88] and was later applied
to recognize gestures [28, 37] requires that both spatial and temporal segmentation be done as a preprocessing step. Specifically, DTW assumes that a single feature vector can be reliably extracted from every input frame. However, this assumption is often hard to satisfy in vision-based systems, where the gesturing hand cannot be located with absolute confidence. Furthermore, DTW assumes that the start and end point of the input gesture are known. This assumption is hard to satisfy in online recognition systems, where the start and end points are not known and cannot be easily inferred.

The framework proposed in this thesis can spot gestures in the absence of spatial or temporal segmentation. The method can accommodate multiple detections of candidate hand regions at every time instance. The method associates a particular model with the best sequence of candidate hand regions. This is similar to other data association problems solved by multiple hypothesis trackers [90] and CONDENSATION-based trackers [51], but in contrast to those which are used for tracking, the method proposed in this thesis is used for recognition, and in fact, no explicit tracking is performed. The work most related to ours is by Sato and Kobayashi [96], who extended the Viterbi algorithm of hidden Markov models (HMMs) so that multiple candidate observations can be accommodated at every input frame. Our method differs from [96] in that it is extended further to continuous gesture recognition, is evaluated in a more challenging setting (users are wearing short sleeve shirts and the hand is not an isolated skin-colored blob), and can handle two-handed gestures.

To recognize gestures in continuous video streams, detection of the gesture boundaries is required, i.e., detection of the start and end frame of each gesture. The task of detecting gesture boundaries is often called gesture spotting. There are two basic approaches for detecting gesture boundaries: the direct approach, which precedes recognition of the gesture class, and the indirect approach, where spotting is intertwined with recognition. Methods that belong to the direct approach first compute low-level motion parameters such as velocity, acceleration, and trajectory curvature [56] or mid-level motion parameters such as human body activity [55], and then look for abrupt changes (e.g., zero-crossings) in those
parameters to identify candidate gesture boundaries. A major limitation of such methods is that they require each gesture to be preceded and followed by non-gesturing intervals – a requirement not satisfied in continuous gesturing, e.g., in American Sign Language (ASL).

Indirect spotting methods detect gesture boundaries by finding, in the input sequence, intervals that give good recognition scores when matched with one of the gesture classes. Most indirect methods [3, 67, 80] are based on extensions of DP algorithms for isolated gestures, e.g., HMMs [102] and DTW [37]. In those methods, the gesture endpoint is detected when the recognition score or likelihood rises above some fixed or adaptive [67] threshold, and the gesture start point can be computed, if needed, by backtracking the optimal DP path. Our method is also an indirect spotting method; however, unlike these past spotting approaches, our method does not require the hand to be unambiguously localized at each frame of the video sequence.

After a provisional set of gesture candidates has been detected, a set of rules is applied to select the best candidate, and to identify the input subsequence with the gesture class of that candidate. Different sets of rules have been proposed in the literature: peak finding rules [75], spotting rules [125], and the user interaction model [129]. A common problem that occurs in practice but is often overlooked is the subgesture problem, i.e., frequent false detection of gestures that are similar to parts of other longer gestures. To address this problem heuristic information, such as moving the hand out of the camera range or freezing the hand for a while, can be used to catch the user’s completion intentions [67]. However, this approach limits the naturalness of the user interaction. In our preliminary work [3], the system was provided with a manual specification of subgesture-supergesture pairs. Unfortunately, manually specifying these pairs does not scale well with the number of gestures, and can be quickly become tedious, or even infeasible. In this thesis we show how subgesture-supergesture pairs can be learned automatically from training data.
2.6 Applications

There are many applications that require spotting gestures in continuous input streams. The main challenge in those applications is how to decide when a gesture starts or ends. When the input is a video stream, then an additional challenge is how to handle situations where the gesture trajectory is difficult to extract. This section describes three applications: handwriting recognition, human computer interaction, and sign language recognition.

Handwriting Recognition

Automatic recognition of handwriting has many practical applications for reading handwritten notes in a personal digital assistants (PDAs), for reading postal addresses on envelopes [66], for reading amounts in bank checks, for verifying signatures in credit card transactions [76], and for spotting words in historical documents [91]. Handwriting recognition algorithms can be grouped into two categories depending on the acquisition method used to obtain the handwriting: offline and online algorithms [86]. Offline methods take as input scanned images of handwritten characters [66, 14], words, or documents [91]. Online methods take as input the samples of a trajectory traced by a stylus or pen [76, 9].

The work presented in this thesis belongs to the second category of online algorithms. Most online systems (e.g., [76, 9]) assume that the trajectories of the characters [9] or words [76] are temporally segmented, that is, the start and end points of the input trajectories are known. The methods used in those systems fall under the category of whole matching methods that were described in Section 2.1. In contrast, the methods presented in this thesis do not assume that temporal segmentation of the input trajectory is known. The goal is to find a pattern or a subsequence within a longer sequence.

The data used to evaluate online handwriting recognition systems (e.g., [9]) are typically acquired with pen-based interfaces that directly record samples of the trajectories traced out by the pen. For example, the data in the widely used UNIPEN benchmark [48] was acquired in this way. In camera-based handwriting recognition interfaces (e.g., [73, 76] the
trajectories are not directly observable. They have to be recovered from image observations.

This presents two challenges: first, the system has to locate and track the gesturing object in the image [76], and second, the system has to segment the word [73] (or signature [76]) to be recognized. In this thesis we address these two problems in a unified way by using a spatiotemporal matching algorithm that can compute a useful matching cost even when the location of the gesture is ambiguous, and a spotting algorithm that can detect when a gesture ends.

Human Computer Interaction

Another successful application of gesture recognition systems is in the area of Human Computer Interaction (HCI). Some systems have been specifically designed to provide computer access for deaf people [50] and people with severe disabilities [16], while other systems have been used to control virtual environments [15], computer applications [67, 121], television sets [42], and video games [41, 56].

For example, Berry et al. [15] used hand gestures for navigating a virtual environment, and for selecting and moving virtual objects in the environment. Ju et al. [54] developed an automatic system for analyzing and annotating video sequences of technical talks. Davis and Bobick [38] implemented a prototype system for a virtual Personal Aerobics Trainer (PAT), that can recognize stretching and aerobic movements, and can even personalize an aerobics session to meet the user’s needs. MacCormick and Isard [70], and Black and Jepson [18] have developed systems that can track hand gestures and recognize drawing on boards. Triesch and Maslburg [110] developed a person-independent gesture interface on a real robot, which allows the user to interact with the robot through the use of commands, such as how to grasp an object and where to put it.

Some vision-based HCI systems have to recognize static hand shapes and gestures, but others have to recognize dynamic gestures from continuous video. Many dynamic gesture recognition systems assume that gesture boundaries can be reliably identified by imposing certain constraints. For example, a distinct hand shape can be used to signal the start or
end of a gesture. Alternatively, the user may be required to not move the hand for a certain period of time, or to move the hand to a specified location, or even out of the camera’s field of view. Such requirements make the interaction very awkward and unnatural. The objective of the spotting framework developed in this thesis is to enable HCI systems, which impose fewer constraints on the user, and which can enable more natural interaction.

**Sign Language Recognition**

Another application that requires gesture spotting and recognition is the analysis of sign language. Automatic analysis of sign language can lead to many useful applications including [82]: sign-to-text or sign-to-speech translation systems, dialog systems for specific transactional domains, bandwidth-conserving communication between signers through the use of animated avatars, and automatic or semiautomatic annotation of sign language video databases. A full understanding of sign language requires the analysis of both manual signals and non-manual signals, conveyed through facial expression, head movements, body postures and torso movements. Generally speaking, lexical information is conveyed through manual signals, while semantics is conveyed through nonmanual signals. Although the methods presented in this thesis are applicable to different types of gestures, we focus on recognizing hand gestures, and therefore survey only those related works. Analysis of nonmanual signals in sign language is thoroughly reviewed in a recent survey [82].

The research in recognition of manual signing can be categorized according to data acquisition, feature extraction and classification methods. Hand gesture data can be acquired using direct-measure (glove-based) devices [116, 117], or using cameras [101, 32, 124]. Generally speaking, glove-based methods are more accurate than vision-based methods. This is a direct consequence of the very nature of the direct measurement process. Instrumented gloves directly measure the hand 3D pose (position and orientation) and shape (joint angles). In contrast, vision-based algorithms aim at recovering (2D or 3D) hand pose and shape from image observations. This is a much harder task that greatly depends on various imaging conditions including clothing, background, field of view, image resolution, and
illumination. While few glove-based methods have been successfully applied to large vocabularies on the order of thousands of signs [117], vision-based methods have been limited to vocabularies of few tens of signs [101, 34, 124].

Vision-based methods for automatic analysis of hand gestures typically follow a bottom-up framework and include three components: hand localization and tracking, feature extraction, and gesture classification. Hand localization and tracking were already discussed in Section 2.3. Here we mainly discuss the assumptions often made in the context of sign language recognition. The hands are typically located using color, motion, and/or edge information [34, 124]. If skin color detection is used, then the signer is required to wear long-sleeved clothing, with restrictions on other skin-colored objects in the background. The hands are often distinguished from the face by assuming that the face is relatively static, and the head region is bigger in size. A face detector can also be used for this purpose. However, many appearance-based face detectors will fail if the hand occludes a large portion of the face. When motion cues are used, the underlying assumption is that the hand is the only moving object on a stationary background and that the other body parts of the signer are relatively static. Another common requirement is that the hand must be constantly moving.

Until recently appearance-based hand detection methods were not very popular mainly due to the typical high computational cost involved. Such methods [81, 63] became more popular following the success of Viola-Jones face detector [112], which is both accurate and efficient. However, the human hand is an articulated object with many degrees of freedom, and has a significantly higher variation in appearance compared to a face. Therefore, current appearance-based methods are limited to a relatively small number of hand shapes and poses.

The basic components of a sign gesture consist of hand shape, hand orientation, location, and motion [106]. Hand shape refers to the finger configuration, hand orientation refers to the direction of the palm, hand location to where the hand is placed relative to the body, and motion to the trajectory traced by the hand. Two different signs may be
similar in all but a single component. There is a limited number of each of the basic components which combine to make up all the possible signs. The exact number depends on the particular sign language and the phonological model adopted, but generally speaking there are only a few tens of hand shapes, locations, and motions, and even fewer hand orientations [106].

Vision-based sign language recognition methods extract visual features that correspond to the abovementioned basic components of a sign. Hand location is often represented by the centroid of the hand region with respect to a body part which can be easily detected [102]. Hand motion is modeled by the blob centroid motion [102] or by pixel motion trajectories [124]. Accurate 3D hand orientation is difficult to extract from images, but a coarse approximation can be estimated from the 2D moments of an ellipse fitted to the hand blob or contour [102]. Hand shape features are often extracted from the hand contour. Fourier descriptors [49] and shape context features [81] have been used to give a global shape description, while contour curvature [6] has been used to detect local finger protrusions. Hand shape descriptors are very sensitive to image noise and scene clutter, and they are very difficult to extract when the hand occupies a small portion of the field of view. In this thesis we focus on designing ASL spotting and recognition methods which make the fewest assumptions about the environment and the imaging conditions, and which will not break down under such conditions.

2.7 Summary

This chapter presented a variety of methods for finding occurrences of patterns in long input streams. Those methods have applications in many different domains. One such application, and the one that we are considering in this thesis, is finding occurrences of gestures in video streams. What makes this problem of gesture spotting particularly challenging compared to other problems is that in addition to finding the start and end points of the gesture (temporal segmentation) the gesture has to be located in every frame of the sequence (spatial segmentation).
Existing methods assume that the gesture can be reliably located in every frame of the video. This is typically done by detecting where the gesturing object (e.g., hand) is in the first frame, and then tracking it from frame to frame. Despite recent advances in visual tracking, the biggest challenge remains to track an object over extended periods of time without tracking failure. Our proposed method can be applied in domains where existing algorithms cannot reliably localize the gesturing object. Instead of assuming perfect localization, we make the milder assumption that a list of candidate object locations is available for each frame of the input sequence. The combined spatiotemporal gesture segmentation method simultaneously finds the start and end point of the gesture of interest, while at the same time it solves a data association problem, namely it identifies the best object location out of the multiple hypotheses available at every frame. While the method is applied to gesture spotting in this thesis, it is general and can be applied to other problems that need to solve a data association problem within a dynamic programming framework for matching, for instance recognizing speech utterances of a particular speaker in multi-speaker environments, or recognizing certain human actions in surveillance applications.

Dynamic programming-based search methods can become too expensive for real-time implementations of systems, particularly when the underlying vocabulary (of e.g., words or gestures) is large. In our case the search time also increases with the number of candidate location hypotheses. Several pruning methods have been proposed that prune unlikely hypotheses and propagate only the most promising hypotheses from one time instance to another. Such methods often employ ad hoc pruning decisions. We propose to view pruning as a binary classification problem. In light of this view, more principled pruning decisions can be made by classifiers that are learned from training data. The objective of the classifiers is to maximize pruning, while at the same time minimize the chance of pruning correct hypotheses. In other words, the objective is to find the best tradeoff between accuracy and efficiency.

Finally, most existing gesture spotting methods will recognize a gesture immediately when its model matching cost becomes sufficiently low. However, when one gesture resem-
bles a part of another longer gesture, then when the longer gesture is performed, the system may falsely detect the occurrence of the shorter gesture. One way to avoid this problem is to design a set of gestures where no gesture contains another gesture, but this is not always possible. For a given set of gestures we propose to learn from training data, a pairwise subgesture relation. A gesture is a subgesture of another longer gesture if it appears as a part of the longer gesture. For a particular gesture we learn the set of gestures that includes it as a subgesture, and use that information to make the correct spotting decision.

The next three chapters of the thesis describe our formulation in greater detail. Chapter 3 introduces the proposed spatiotemporal matching algorithm. Chapter 4 shows how to speed up the matching by using pruning. Finally, Chapter 5 presents the algorithm for spotting a gesture of interest in a video stream.
Chapter 3

Spatiotemporal Matching

This chapter describes the proposed spatiotemporal matching algorithm. The chapter starts by reviewing Dynamic Programming (DP) [13], which is a general method for reducing the runtime of algorithms that have certain properties. All the matching methods described in this chapter make use of dynamic programming. The following section describes Dynamic Time Warping (DTW), which is a method for computing a temporal alignment between two sequences, and a corresponding matching cost. DTW is an algorithm suited for the problem of matching two segmented sequences, also known as whole sequence matching. The extension of DTW to the problem of subsequence matching is called Continuous Dynamic Programming (CDP) and is described next. After the necessary background is laid out, we then describe our spatiotemporal matching algorithm. First, we introduce Dynamic Space Time Warping (DSTW), an extension of DTW for whole sequence matching, which aligns two sequences both in time and space. Finally, we present our spatiotemporal matching algorithm for subsequence matching, which combines the temporal segmentation capability of CDP with the spatial localization capability of DSTW. This algorithm can be generally applied to problems where the input consists of multiple feature vectors at every time instance. Table 3.1 shows a list of symbols that will be used in this chapter, and the rest of the thesis.
<table>
<thead>
<tr>
<th>Variable Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>time index for model features or index for model states</td>
</tr>
<tr>
<td>$j$</td>
<td>time index for input features</td>
</tr>
<tr>
<td>$k$</td>
<td>candidate index for multiple input feature vectors</td>
</tr>
<tr>
<td>$K$</td>
<td>number of candidate input feature vectors per time instance</td>
</tr>
<tr>
<td>$m$</td>
<td>number of model features or model states</td>
</tr>
<tr>
<td>$n$</td>
<td>number of input features</td>
</tr>
<tr>
<td>$g$</td>
<td>index for gesture model or gesture example</td>
</tr>
<tr>
<td>$M$</td>
<td>model</td>
</tr>
<tr>
<td>$Q$</td>
<td>input or query</td>
</tr>
<tr>
<td>$M_i$</td>
<td>model feature vector at time $i$</td>
</tr>
<tr>
<td>$Q_{jk}$</td>
<td>input feature vector $k$ at time $j$</td>
</tr>
<tr>
<td>$d(i,j,k) \equiv d(M_i, Q_{jk})$</td>
<td>local matching cost between model feature vector $M_i$ and input feature vector $Q_{jk}$</td>
</tr>
<tr>
<td>$W$</td>
<td>warping path that defines temporal alignment between input subsequence and model subsequence</td>
</tr>
<tr>
<td>$D(W)$</td>
<td>cumulative cost of a warping path</td>
</tr>
<tr>
<td>$W^<em>, D^</em>$</td>
<td>optimal warping path and corresponding optimal cost</td>
</tr>
<tr>
<td>$T$</td>
<td>training set</td>
</tr>
<tr>
<td>$V$</td>
<td>validation set</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>expansion factor</td>
</tr>
<tr>
<td>$C$</td>
<td>classifier</td>
</tr>
<tr>
<td>$C_i(Q_{jk})$</td>
<td>output of classifier $i$ on input feature vector $Q_{jk}$</td>
</tr>
<tr>
<td>$node$</td>
<td>dynamic programming table node</td>
</tr>
<tr>
<td>$list$</td>
<td>list of active nodes</td>
</tr>
<tr>
<td>$ind$</td>
<td>array of pointers to list of active nodes</td>
</tr>
<tr>
<td>$ST$</td>
<td>subgesture table</td>
</tr>
</tbody>
</table>

**Table 3.1:** Table of Symbols.

### 3.1 Dynamic Programming

The spatiotemporal matching algorithm proposed in this thesis uses dynamic programming to compute, for every model, an alignment between the input and the given model, and a corresponding matching cost. Dynamic programming (DP) [13] is a general approach for reducing the runtime of algorithms exhibiting two properties: *optimal substructure* and *overlapping subproblems* [27].

Optimal substructure means that optimal solutions of subproblems can be used to find
the optimal solutions of the overall problem. When a problem exhibits optimal substructure we can solve it using a three step divide and conquer approach:

1. Break the problem into smaller subproblems.

2. Solve these subproblems using this three step approach recursively.

3. Use these optimal solutions to construct an optimal solution for the original problem.

A problem is said to have overlapping subproblems if the problem can be broken down into subproblems, which can be reused several times.

All the matching algorithms described in this chapter exhibit both of these properties. The original problem is computing a matching cost between two subsequences. The general idea is that this problem can be broken down into subproblems of computing matching costs between shorter subsequences. The matching costs computed for these subproblems can then be combined to compute the optimal matching cost for the original problem.

3.2 Dynamic Time Warping (DTW)

Dynamic Time Warping [64] is used to compute a distance between two time series. The time series is a list of samples taken from a signal, ordered by the time that the respective samples were obtained. The time series can be a simple string of characters drawn from a finite alphabet, but we will assume for the rest of the chapter that it is a real-valued multidimensional signal sampled in time, e.g., the 2D position and velocity of the hand.

A naive approach to calculating a matching distance between two time series can be to resample one of them or both, such that both time series will have the same number of samples, and then compare the series sample by sample. If the original time series are similar in shape but have different temporal scales (e.g., they represent gestures which are performed at different speeds), then the naive approach may compare samples that might not correspond well.

Dynamic Time Warping finds the optimal alignment between samples in the two time series. The alignment is optimal in the sense that it minimizes the cumulative distance
measure which is the sum of all local distances between aligned samples. The procedure is called time warping because it warps the time axes of the two time series in such a way that corresponding samples appear at the same location on a common time axis. The two time axes are aligned, in general, in a non-linear way.

Let $M$ and $Q$ be two time series of length $n$ and $m$, respectively, where

\[ M = (M_1, M_2, \ldots, M_i, \ldots, M_m) \]  
\[ Q = (Q_1, Q_2, \ldots, Q_j, \ldots, Q_n). \]

The DTW distance between the two sequences $M$ and $Q$ is computed using dynamic programming. We construct an $m$-by-$n$ matrix where each element $(i, j)$ contains the local distance $d(M_i, Q_j)$ between the samples $M_i$ and $Q_j$. A warping path $W$ is a set of matrix elements $w_t = (i_t, j_t)$ that defines an alignment between $M$ and $Q$. Formally,

\[ W = w_1, w_2, \ldots, w_T, \ max(m, n) \leq T \leq m + n - 1 \]

The warping path is subject to several constraints [58]:

- **Boundary conditions:** $w_1 = (1, 1)$ and $w_T = (m, n)$. This requires the warping path to start by matching the first frame of the model with the first frame of the query, and end by matching the last frame of the model with the last frame of the query.

- **Continuity:** Given $w_t = (a, b)$ then $w_{t-1} = (a', b')$, where $a - a' \leq 1$ and $b - b' \leq 1$. This restricts the allowable steps in the warping path to adjacent cells.

- **Monotonicity:** Given $w_t = (a, b)$ then $w_{t-1} = (a', b')$ where $a - a' \geq 0$ and $b - b' \geq 0$. This forces the warping path sequence to increase monotonically in time.
Dynamic Time Warping finds the path that minimizes the warping path cost:

\[ DTW(M, Q) = \min_W \sum_{t=1}^{T} d(w_t). \]  

(3.4)

A brute force method for finding the minimum warping path cost requires the evaluation of an exponential number of warping paths. Fortunately, dynamic programming can be used to find the minimum warping path cost in \( O(mn) \) time, by defining the cumulative distance \( D(i, j) \) recursively:

\[ D(i, j) = \min\{D(i-1, j-1), D(i-1, j), D(i, j-1)\} + d(i, j), \]  

(3.5)

with the boundary conditions

\[ D(0, 0) = 0, \quad D(0, j) = D(i, 0) = \infty. \]  

(3.6)

The DTW distance is simply

\[ DTW(M, Q) = D(m, n). \]  

(3.7)

The algorithm starts with computing the matching cost \( D(1, 1) \). \( D(1, 1) \) can be computed from the solutions to the subproblems \( D(0, 0) = 0, \ D(0, 1) = \infty, \) and \( D(1, 0) = \infty. \) The minimum of 0 is added to the local matching cost between the samples \( M_1 \) and \( Q_1 \). The algorithm then proceeds from top to bottom with computing \( D(2, 1), D(3, 1), \ldots, D(m, 1) \). Then the algorithm proceeds to the next column of the table, column 2, and computes \( D(1, 2) \). The algorithm continues scanning the table left-to-right and top-to-bottom, until the last column and row are reached and \( D(m, n) \) is computed (see Figure 3·1). By definition \( D(m, n) \) is the DTW distance between the sequences \( M \) and \( Q \).

We note that other definitions for the warping path and the associated recursion are possible. The definitions used above are the most widely used, and lead to the symmetric
Figure 3-1: The Dynamic Time Warping (DTW) algorithm. The algorithm starts at cell (1, 1) and proceeds top to bottom and left to right updating the cumulative distance $D(i, j)$ at every cell using the cumulative distances of neighboring cells. The optimal matching cost (DTW distance) between the model pattern $M$ and the input pattern $Q$ is $D(m, n)$. The optimal warping path $W$, which is represented by the dashed line and shaded cells, can be recovered by tracing back the sequence of optimal neighbors. $W$ provides information about the optimal correspondence (alignment) between the input features and the model.

DTW distance (i.e., $D(M, Q) = D(Q, M)$).

For the purpose of simplifying notation later on in the section we define the neighborhood of the current cell $w_t = (i, j)$ as the set $N(i, j)$ of all valid predecessor cells. Those cells are valid in the sense that they obey the warping path constraint. In the case of the symmetric DTW the neighborhood is defined as follows:

$$N(w_t) \equiv N(i, j) = \{(i - 1, j - 1), (i - 1, j), (i, j - 1)\}. \quad (3.8)$$
3.3 Continuous Dynamic Programming (CDP)

In this thesis, we are interested in spotting patterns in video streams. The DTW distance is well suited for whole sequence matching, where the matching is between two sequences $M$ and $Q$ that are temporally segmented. For example, in the context of gesture recognition, $M$ and $Q$ can be two segmented gestures, with known start and end points, and DTW computes a matching cost between those gestures. However, in many practical applications one of the sequences, say $Q$, can be a long sequence or even a continuous stream, and the goal is to find occurrences of the sequence $M$ in $Q$. This problem is known as subsequence matching.

A naive approach to solve the subsequence matching problem would be to compute the DTW matching cost between the sequence $M$ and all possible subsequences of $Q$, and find those subsequences with small matching costs. However, under nonlinear time warping, the length of a candidate subsequence can be arbitrary in general, and therefore all subsequences formed from all possible start and end points have to be evaluated.

Continuous dynamic programming (CDP) [80] is an extension of DTW that can solve the subsequence matching problem in a single forward pass over the input $Q$. CDP augments the model $M$ with a dummy state $M_0$ that matches every feature of the input $Q_j$ perfectly, i.e., $D(0, j) = 0, \forall j$. The algorithm can thus generate a new candidate optimal warping path at every time instance.

DTW is typically executed in an offline mode. Therefore usually the entire DP table for a particular model can be stored in memory. In contrast, CDP is typically executed in an online mode, and therefore some restrictions on the memory requirement have to be applied. Equation 3.5 suggests that for a particular model only the previous column $j - 1$ and the current column $j$ of that model’s DP table have to be stored in memory in order to update the cumulative costs of all DP cells of the current column. This is true for the specific warping path constraints implicit in Equation 3.5. However, often some post-matching procedures may require knowledge about the complete optimal warping path,
which can be recovered by tracing back the sequence of optimal neighbors. Therefore, it is common practice to allocate for each model a queue (or a circular buffer) of fixed size. Given a new input frame $j$, the current column is appended to the tail of the queue, and the least recent frame is removed from the head of the queue. The queue is essentially an implementation of the sliding window concept for subsequence matching. The size of the queue should be to a value, which is greater or equal to the longest expected gesture duration.

CDP computes at every frame $j$ the cumulative matching cost $D(m, j)$. The question that remains is how to decide when the matching cost is small enough to determine that a subsequence has just been detected. The answer to this question is deferred to Chapter 5.

### 3.4 Dynamic Space Time Warping (DSTW)

The Dynamic Time Warping algorithm and its extension to continuous input, the Continuous Dynamic Programming algorithm, assume that a single feature vector can be reliably extracted from the input at every time instance. However in practice, this assumption is often hard to guarantee. For example, when the input is an image sequence, and the goal is to find the occurrence of a dynamic hand gesture then for every image in the sequence a feature vector will be computed from an image patch that corresponds to the hand. However, detecting a hand in an image is a very challenging task, which can occasionally return an image patch that does not correspond to the hand. A feature vector that is computed from an incorrect image patch can lead to a very poor match.

If instead of detecting a single image patch we detect multiple image patches, such that one of these patches corresponds to the hand, then multiple feature vectors can be computed, one of which corresponds to the correct hand image patch. The goal of the matching algorithm would be to compute the matching cost between the input and model and at the same time identify the correct feature vector in every time instance.

This section presents Dynamic Space Time Warping (DSTW) which similar to DTW computes a distance between a model time series $M$ and an input time series $Q$. However,
now each $Q_j$ is not a single feature vector but a set of feature vectors, one of which corresponds to the “correct” observation. Formally, let $M$ and $Q$ be two time series as defined in Equations 3.1 and 3.2. Let each $Q_j$ be a set of feature vectors: $Q_j = \{Q_{j1}, \ldots, Q_{jK}\}$, where each $Q_{jk}$, for $k \in \{1, \ldots, K\}$, is a feature vector, and $K$ is the number of feature vectors extracted from each query frame. $K$ can be either fixed or vary from frame to frame.

The DSTW distance between the two sequences $M$ and $Q$ is computed using dynamic programming. Conceptually, we construct a 3D matrix of size $m$-by-$n$-by-$K$, where each element $(i,j,k)$ contains the local distance $d(M_i, Q_{jk})$ between the model state $M_i$ and the feature vector $Q_{jk}$.

The warping path $W$ is defined in a similar fashion, but now each element $w_t = (i,j,k)$ is a triple specifying that model state $M_i$ is matched with feature vector $Q_{jk}$. We say that $w_t$ has two temporal dimensions (denoted by $i$ and $j$) and one spatial dimension (denoted by $k$). The modified warping path constraints are:

- **Boundary conditions:** $w_1 = (1,1,k)$ and $w_T = (m,n,k')$. This requires the warping path to start by matching the first frame of the model with the first frame of the query, and end by matching the last frame of the model with the last frame of the query. No restrictions are placed on $k$ and $k'$, which can take any value from 1 to $K$.

- **Temporal continuity:** Given $w_t = (a, bk)$ then $w_{t-1} = (a', b', k')$, where $a - a' \leq 1$ and $b - b' \leq 1$. This restricts the allowable steps in the warping path to adjacent cells along the two temporal dimensions.

- **Temporal monotonicity:** Given $w_t = (a, b, k)$ then $w_{t-1} = (a', b', k')$ where $a - a' \geq 0$ and $b - b' \geq 0$. This forces the warping path sequence to increase monotonically in the two temporal dimensions.

Note that continuity and monotonicity are required only in the temporal dimensions. No such restrictions are needed for the spatial dimension; the warping path can “jump” from any spatial candidate $k$ to any other spatial candidate $k'$. 
Given warping path element $w_t = (i, j, k)$, we define the set of neighbors $N(w_t)$ to be the set of all possible predecessors $w_{t-1}$ that satisfy the warping path constraints (in particular continuity and monotonicity):

$$N(w_t) \equiv N(i, j, k) = \{(i, j - 1), (i - 1, j - 1)\} \times \{1, \ldots, K\} \cup \{(i - 1, j, k)\}. \quad (3.9)$$

An example neighborhood for $K = 2$ is depicted in Figure 3·2. In general, the size of the set of neighbors $N(w_t)$ is $2K + 1$. The reason we include only one candidate $(i - 1, j, k)$ from the previous state $i - 1$ and the current input frame $j$ is that it does not make sense to consider two different candidates $k$ and $k'$ for the same input frame $j$, while changing from state $i - 1$ to state $i$. If in addition we do not allow two different states $i - 1$ and $i$ to match the same input feature $j$, as is the case in HMMs, then the neighborhood size becomes $2K$.

Dynamic Space Time Warping finds the path that minimizes the warping path cost:

$$DSTW(M, Q) = \min_{W} \sum_{t=1}^{T} d(w_t), \quad (3.10)$$
with the modified definitions of \( W, w_t, \) and \( d. \) The corresponding definition of the cumulative distance \( D(w_t) \equiv D(i, j, k) \) is:

\[
D(w_t) = \min \{ D(N(w_t)) \} + d(w_t),
\]

with the boundary conditions

\[
D(0, 0, 0) = 0, D(0, j, k) = D(i, 0, k) = \infty.
\]

The DSTW distance is simply

\[
DSTW(M, Q) = \min_k D(m, n, k).
\]

### 3.5 Translation Invariance in DSTW

In recognizing hand gestures, a commonly used feature is position of the hand. Using positions as features is appealing because they are simple to extract, and are highly informative about the gesture content. However, they are not invariant to translation. In simple DTW, there is a single candidate per frame, and therefore the trajectory of the hand is known. In that case we can achieve translation invariance (i.e., invariance with respect to global translation of the entire gesture) by subtracting from the entire trajectory the position of the hand in the first frame.

In DSTW, we can apply this strategy to the model, where the trajectory is known. However, for the input sequence, there are multiple candidates at each frame and therefore the position of the hand in the first frame is not known. When position is used as a feature, we can achieve translation invariance as follows: given the \( K \) candidate regions detected at the first frame, we run \( K \) separate DSTW matching processes, running in parallel. Each such process \( P_k \) corresponds to a candidate \( k \) among the \( K \) regions detected in the first frame. The process \( P_k \) makes the assumption that \( k \) was the correct candidate in the first
frame, and normalizes all position features in subsequent frames by subtracting from them the position of the $k^{th}$ candidate in the first frame. When all frames have been processed, to find the best match of the observation sequence with the model, we need to find which of the $K$ parallel DSTW processes gave the lowest matching cost $D^*$.

### 3.6 Combining DSTW and CDP

The spatiotemporal matching method described in Algorithm 1 combines ideas from Dynamic Space Time Warping (DSTW) [5] and Continuous Dynamic Programming (CDP) [80], and inherits the advantages of these two methods: as in DSTW, the method allows the input sequence to be matched to the model in the absence of precise hand localization at each frame; it is sufficient that the true hand location be included in the list of $K$ candidates for that frame. As in CDP, the method does not require that the start and end frame of the gesture be known in advance. The proposed algorithm can be applied in settings where neither spatial nor temporal segmentation can be obtained via preprocessing. This is in contrast to existing gesture recognition methods, including DSTW and CDP, which assume that at least one of the two segmentation problems can be handled during preprocessing.

The spatiotemporal matching algorithm finds for a given input frame $j$ the optimal warping path $W^*$, which is simply the warping path that minimizes the cumulative matching cost. The optimal path $W^* = (i_1^*, j_1^*, k_1^*), \ldots, (i_T^*, j_T^*, k_T^*)$ provides three important pieces of information:

- $D(W^*)$ is a measure of how well the input sequence matches the model. $D(W^*)$ can be used to determine if the gesture modeled by $M$ has actually occurred in the input sequence $Q$. If we have multiple gesture classes, comparing the matching scores between the input sequence and each class model the system can be used in deciding which model provides the best match, and thus enabling the system to classify the gesture.
• $W^*$ specifies, for each input frame, the candidate hand location that optimizes the matching between the input sequence and the model. In other words, $W^*$ specifies the optimal spatial segmentation of the gesture.

• $W^*$ specifies the optimal temporal segmentation of the gesture. The starting frame of the gesture is the first $j_t^*$ such that $i_t^* = 0$, i.e., the first input frame that is not matched to the dummy state $M_0$. The end frame of the gesture is $j_T^*$, i.e., the last input frame matched to the model according to $W^*$.

Finding the optimal warping path $W^*$ between the input sequence and a set of models is equivalent to simultaneously classifying the input sequence and identifying the best hypothesis for spatial and temporal gesture segmentation. The spatiotemporal segmentation obtained this way uses not only low-level information, but also high-level information obtained by matching the input sequence with a gesture model. As will be seen in the experiments, using this additional high-level information makes it possible to reliably segment and recognize gestures even in cases where low-level information by itself is insufficient for reliable gesture segmentation.

The pseudocode for the spatiotemporal matching method is given in Algorithm 1. The block of Lines 1-5 is executed in the beginning of the algorithm, or after a gesture has been spotted. This block initializes or resets the dynamic programming table in such a way that the update performed in Line 13 can be applied to all input frames $j$, including $j = 1$. Lines 6-8 initialize, for all candidates, the dummy state $i$ to a cumulative distance value of 0. Thus, the dummy state matches all the input feature vectors $Q_{jk}$ perfectly, thereby allowing for a new warping path to be generated at every frame $j$ from any candidate $k$. Lines 9-18 constitute the main part of the algorithm, where the dynamic programming update occurs. Line 15, which corresponds to Equation 3.11, updates the cumulative distance of the current cell, and Line 16 records the optimal neighbor of the current cell in $b$. The optimal neighbors can be used to reconstruct the optimal path in Lines 21-25. Line 20 records the optimal matching cost $D^*$. Finally, Lines 21-25 can be optionally executed
to construct the optimal warping path $W^*$. If the end frame of a gesture has been detected at a particular frame $j_e$, then the corresponding start frame $j_s$ can be determined from $W^*$.

The description in Algorithm 1 is given for matching the input with a single model. This is useful for finding one particular gesture of interest in the input. However if the goal is to find the occurrences of multiple gestures, then Algorithm 1 is used to compute the matching scores between the input and all the models. The optimal matching costs $D^*_g = D(m_g, j, k^*_g)$ for each model are then fed into the gesture spotting and recognition module, to decide whether or not a gesture has ended at frame $j$ (spotting), and if a gesture was spotted then decide to which class it belongs (recognition). The spotting and recognition algorithms are described in Chapter 5.

3.7 Matching with Multiple Input Sources

In the previous sections it was assumed that the input comes from a single source, for example, that a gesture is performed by a single hand. The problem was to associate the model with the correct candidate in every frame. The extension to multiple input sources can be done in a number of ways. For illustration purposes, we will limit our discussion to $N = 2$ input sources, for example, a gesture that is performed using two hands.

One way to model two-handed gestures is to concatenate the observation vectors of the two hands maintaining a consistent order, namely, first the feature vector of the left hand followed by the feature vector of the right hand [101]. Thus, each model feature $M_i$ is a concatenation $M_i \equiv (M^l_i, M^r_i)$ of the the left and right hand features. During the online recognition phase the hand identities are determined in the first frame, and maintained by tracking the color blobs of each hand independently [101]. Therefore, an input feature $Q_j$ could also be formed by a concatenation $Q_j \equiv (Q^l_j, Q^r_j)$ of the left and right hand features, and matched with the different model states. If one of the hands occludes the other, resulting in a single colored blob, then a single feature vector is extracted from that blob, and that feature vector is duplicated for both hands, i.e., $Q^l_j = Q^r_j$. 
Algorithm 1: A Spatiotemporal matching algorithm for finding the optimal warping path between an input sequence and a gesture model.

**input**: A sequence of model feature vectors $M_i, 0 \leq i \leq m$, input frame $j$, and a set of input feature vectors $Q_j = \{Q_{j1}, \ldots, Q_{jK}\}$.

**output**: A global matching cost $D^*$, and an optimal warping path $W^* = (w^*_1, \ldots, w^*_T)$.

// Initialization of the cumulative matching cost. This block is executed only for $j = 1$, or after a subsequence has been spotted in the previous frame.

1. for $i = 0 : m$
2.   for $k = 1 : K$
3.     $D(i, j - 1, k) = \infty$
4.   end
5. end

// Iteration: for all $j$.
// First initialize "dummy" state $i = 0$, which enables generation of a new warping path.
6. for $k = 1 : K$
7.   $D(0, j, k) = 0$
8. end

// Loop over all model feature vectors $i$ and all input feature vector $k$ in the current frame
9. for $i = 1 : m$
10.   for $k = 1 : K$
11.     $w = (i, j, k)$
12.       // Consider the matching costs of all the neighbors $w'$ of the current cell $w$.
13.       for $w' \in N(w)$
14.         $C(w', w) = D(w') + d(i, j, k)$
15.       end
16.     // Find the best matching cost $D$ and corresponding neighbor $b$.
17.     $D(w) = \min_{w' \in N(w)} C(w', w)$
18.     $b(w) = \arg\min_{w' \in N(w)} C(w', w)$
19.   end

// Termination: the best matching cost for the input subsequence ending at frame $j$ is stored in $D(m, j, k^*)$.
20. $k^* = \arg\min_k \{D(m, j, k)\}$
21. $D^* = D(m, j, k^*)$

// Backtracking (optional): construct the optimal warping path $W^*$
22. for $t \in \{t^* - 1, \ldots, 1\}$
23.   $w^*_t = b(w^*_{t+1})$
24. end
25. $W^* = (w^*_1, \ldots, w^*_T)$
Using spatiotemporal matching, the model features $M_i$ can be concatenated as before, namely $M_i \equiv (M_l^i, M_r^i)$. However, during the online recognition phase the identities of the hands are not known in the first frame, and furthermore multiple ($K \geq 2$) hand hypotheses are maintained throughout the input sequence. The number of candidate (concatenated) feature vectors will increase to a maximum of $K^2$ reflecting all ordered combinations of left/right hand hypotheses: $Q_{jk} \equiv (Q_{j,k_1}^l, Q_{j,k_2}^r)$, where $1 \leq k \leq K^2$, and $1 \leq k_1, k_2 \leq K$. We note that $K$ out of those $K^2$ combinations that correspond to duplication of a candidate can be discarded for a particular model if the corresponding two handed gesture does not involve hand occlusion.

Another alternative is to have two separate models for the two hands similar to the Parallel HMM model [116]. The advantage of this model over the previous one is that the time and space complexity is linear in the number of sources, i.e., $NK$, rather than exponential, i.e, $K^N$. At the same time, for the parallel model the matching costs of the different models have to be combined in some way, and this is often done in an ad hoc way. In our experiments of spotting two-handed ASL signs we used the first approach. For a small number of input sources $N = 2$ and a relatively small number of candidates $K = 3$ the time penalty is not too high.

### 3.8 Complexity

The time complexity of the matching algorithms presented in this chapter is proportional to the number of cells that are visited by the dynamic programming method. DTW takes $O(mn)$ time, where $m$ and $n$ are the model and input length respectively. DSTW takes $O(Kmn)$ time, where $K$ is the number of input feature vectors in every time instance. The translation invariant version of DSTW takes $O(K^2mn)$ time. DSTW with $N$ input sources takes $O(K^Nmn)$ using the brute force approach or $O(KNmn)$ using the “parallel” approach. All the continuous versions of the matching algorithms have a similar run time behavior. The next chapter describes how to speed up the matching process by using pruning.
The space complexity of the matching algorithms is similar to the time complexity if the algorithms are run in offline mode. In online mode only the current and previous columns (or matrices in the case of DSTW) are needed to be kept in memory, and therefore the space complexity of each of the matching algorithms is a factor of $n$ less than the corresponding time complexity. Alternatively, a circular buffer of a fixed size can be kept for every model to allow the system to trace back the optimal path.
Chapter 4

Pruning

In optimization problems, pruning is used to speed up the search for an optimal solution. This chapter describes how to prune unlikely hypotheses from the dynamic programming search employed in the spatiotemporal matching algorithm, which was described in the previous chapter. The proposed pruning method is general, and can be applied to speed up many sequence alignment problems, which occur frequently in a number of areas including molecular biology, information retrieval, speech recognition, and computer vision.

Many of the methods used to solve sequence matching or shape registration problems employ similarity measures or dynamic models, which make use of dynamic programming to simultaneously find the alignment between a pair of sequences, or between a sequence and a model, and compute a corresponding matching score. For example, the Dynamic Time Warping distance [64], the Edit distance [68], and the Longest Common Subsequence [47] compute the optimal matching cost between a pair of sequences by finding the optimal alignment in a dynamic programming table of size $O(mn)$, where $m$ and $n$ are the lengths of the two sequences. Similarly, hidden Markov models [89], Conditional Random Fields [65], Dynamic Bayesian Networks [46], and other statistical dynamic models compute the optimal alignment between a sequence and the model by means of a trellis (a sort of “unfolding” of the dynamic model states through time) in time proportional to the sequence length and quadratic in the number of model states.

Our learning-based pruning method can be applied to such dynamic programming-based matching methods by learning pruning classifiers from training data off-line, and then using the classifiers in the online matching stage to prune unlikely matching hypotheses.
A popular technique for pruning dynamic programming-based search methods is the beam search \[95\]. Beam search is a heuristic search algorithm that uses a function to estimate the promise of each node or cell it examines during search. The standard beam search unfolds \( b \) of the most promising nodes at each depth of the search, where \( b \) is a fixed number, the beam width. In addition to speeding up the search this also bounds the amount of space required from the search procedure. In another variant of beam search, which is more related to our proposed pruning method, the beam width is a cost value, and beam search unfolds only those nodes that have costs that are within the beam width from the cost of the current optimal hypothesis \[52\].

The key novelty in our proposed pruning method is that we view pruning as a binary classification (or decision) problem. Based on the outcome of the binary pruning classifier we can decide whether to prune or maintain a particular search hypothesis. In other words, for a particular pruning classifier the objective of the learning algorithm is to maximize the chance of pruning incorrect hypotheses and at the same to minimize the chance of pruning correct hypotheses. If the parameters of the pruning classifier are learned from only the training examples, then the resulting classifier may be “tuned” to that particular set of training examples, and as a result it may falsely prune correct hypotheses of novel test examples. In order to avoid this situation, and learn a more conservative pruning classifier, which will better generalize to yet unseen test examples we make use of cross-validation \[62\].

Cross-validation is a method for evaluating a statistical model, a classifier, or an algorithm that has free parameters. In cross-validation, one divides the training data into several parts, and in turn uses one part (the validation set for this round) to test the procedure fitted to the remaining parts (the training set for this round). In the context of pruning classifiers the goal of the validation set is to test how conservative or aggressive the classifier parameter settings should be.
4.1 Pruning Classifiers

Let us consider the spatiotemporal matching algorithm of Chapter 3. Algorithm 1 computes, for each \((i,j,k)\), the optimal warping path ending at \((i,j,k)\). However, there are cases where feature vector \(Q_{jk}\) matches state \(M_i\) so badly, that we can safely prune out the triple \((i,j,k)\), meaning that we reject all warping paths going through \((i,j,k)\). How can the system decide when it is safe to prune out the triple \((i,j,k)\)? Following our past work [3], the decision can be made by a pruning classifier \(C_i(Q_{jk})\). In the simplest possible form, classifier \(C_i(Q_{jk})\) simply checks whether the local matching cost \(d(i,j,k)\) exceeds a threshold \(\tau(i)\) associated with state \(M_i\). That threshold can be manually specified or learned from the training data [3]. The method described in [3] guarantees that none of the training examples, when embedded in a longer sequence, would be falsely pruned. However, the classifiers thus learned will not generalize well to yet unseen test examples. In Section 4.3 we show how to learn pruning classifiers with the objective of minimizing the expected number of false pruning of yet unseen test examples, using cross-validation.

To the best of our knowledge, the concept of model dependent classifiers \(C_i\), which are learned from training data off-line, and are used for pruning during online spotting is novel. Different types of classifiers can be used including: subsequence classifiers, which prune based on the compatibility between an input prefix and a model prefix; single observation classifiers, which prune based on the compatibility between a single observation and a single model state (or equivalently, based on the likelihood of the current observation); and transition classifiers, which prune based on the compatibility between successive observations. An example subsequence classifier can be defined as follows:

\[
C_i(Q_{jk}) = \begin{cases} 
1 & \text{if } D(i,j,k) \leq \tau(i) \\
0 & \text{otherwise}
\end{cases},
\]

where \(D(i,j,k)\) is the cumulative matching cost between model state sequence \(M_{1:i}\) and observation subsequence \(Q_{j',k_j}, \ldots, Q_{j,k_j}\) for some \(j' < j\), and each \(\tau(i)\) defines a decision
stump classifier for model state $M_i$. This classifier will prune hypotheses with significantly high matching cost.

An example observation classifier can be defined as follows:

$$C_i(Q_{jk}) = \begin{cases} 
1 & \text{if } d(i,j,k) \leq \tau(i) \\
0 & \text{otherwise} 
\end{cases}, \quad (4.2)$$

where $d(i,j,k)$ is the local matching cost between the observation $Q_{jk}$ and model state $M_i$, and each $\tau(i)$ defines a decision stump classifier for model state $M_i$. This classifier models the consistency between a single observation and a corresponding model state. It will prune hypotheses where the matching between an observation and model state pair is poor.

An example transition classifier can be defined such that it does not depend on the model state $i$, but it depends instead on two successive observations $j-1$ and $j$. For example, a classifier can be defined as follows:

$$C(Q_{jk}, Q_{j-1,k_{j-1}}) = \begin{cases} 
1 & \text{if } d_E(Q^p_{jk}, Q^p_{j-1,k_{j-1}}) \leq \tau \\
0 & \text{otherwise} 
\end{cases}, \quad (4.3)$$

where $d_E$ is the Euclidean distance, and $p$ represents the part of the feature vector that corresponds to 2D position. This transition classifier models motion continuity. In other words, it models the fact that the moving object cannot move too rapidly in a single time step. In the context of hand gestures, the hand cannot move more than a certain number of pixels between successive frames.

As another example of a transition classifier that depends on two successive observations, a classifier can be defined such that it measure the compatibility between the computed flow direction and the direction of displacement between two successive detec-
\[ C(Q_{j,k}, Q_{j-1,k_{j-1}}) = \begin{cases} 1 & \text{if } \cos(Q_{j-1,k_{j-1}} - \theta(Q_{j-1,k_{j-1}}, Q_{j,k})) \leq \tau \\ 0 & \text{otherwise} \end{cases}, \quad (4.4) \]

where \( Q_{j-1,k_{j-1}} \) is the computed flow direction and \( \theta(Q_{j-1,k_{j-1}}, Q_{j,k}) \) is the direction of the vector connecting two successive detection windows. This classifier can be used to model motion consistency. When matching with a single candidate the direction can be estimated by the direction of displacement between two successive detections, without the need to compute optical flow, since it is assumed that the two detections correspond to the same object, and that the object moved between those two successive detections. The estimated direction can then be compared to the model state direction. This will result in a single observation classifier as before, where instead of using a position feature, a directional feature is used instead. When matching with multiple candidates, and considering a pair of successive windows, besides the fact that they do not necessarily correspond to the same object, one of the corresponding objects or both may be relatively static, and so using the direction between them is meaningless. The proposed classifier can reveal this discrepancy by comparing this direction to the optical flow direction.

The different pruning classifiers can be used separately or combined using any classifier combination method. One alternative is to view the different pruning classifiers as weak classifiers and learn how to linearly combine them into a strong pruning classifier using the AdaBoost algorithm [97]. In our implementation, we combine the pruning classifiers by simply “and”ing the output values of the different classifiers and use the result to make a pruning decision.

### 4.2 Spatiotemporal Matching with Pruning

The CDP matching algorithm described in Section 3.3 computes the cumulative distance \( D(i,j) \) (Equation 3.5) for every dynamic programming cell \((i,j)\). A key observation is that
for many combinations of $i$ and $j$, either the feature-based distance $d(i, j)$ or the cumulative distance $D(i, j)$ can be sufficiently large to rule out all alignments going through cell $(i, j)$. Our contribution is that we generalize this pruning strategy by introducing a set of binary pruning classifiers that can be learned from training data off-line. Those pruning classifiers can then be used to prune certain alignment hypotheses during online matching.

This section describes how the pruning classifiers can be incorporated in the spatiotemporal matching Algorithm 1 in order to speed it up. We call the resulting algorithm Continuous Dynamic Programming with Pruning (CDPP). In order to simplify the notation of the algorithm, we will assume only a single candidate and remove any dependence on the candidate index $k$. CDPP can be easily extended to the case where multiple candidate feature vectors are available at every time instance.

The input to the algorithm is the model $M$, a set of model-dependent pruning classifiers $C_i$, the current input frame $j$, the current input feature vector $Q_j$, and the previous column $col_{j-1}$. We note, that in the online version of the algorithm the local distance $d(i, j) = d(M_i, Q_j)$ and the cumulative distance $D(i, j)$ need not be stored as matrices in memory; instead they can be computed on the fly. It suffices to store for each model (assuming backtracking is not required) only the previous column vector $col_{j-1}$ corresponding to input frame $j - 1$, and then update the current column vector $col_j$ corresponding to input frame $j$.

The basic idea of CDPP is depicted in Figure 4-1. There are two cases of the algorithm corresponding to the two possible outcomes of the binary pruning classifier. The first case corresponds to no pruning: if the pruning classifier $C_i(Q_j)$ outputs 1 (Figure 4-1(a)) then the algorithm simply evaluates Equation 3.5, updates the current cell $(i, j)$, and advances to the next cell $(i + 1, j)$ in the current column vector $col_j$. The second case corresponds to pruning: if the pruning classifier $C_i(Q_j)$ outputs 0 (Figure 4-1(b)) then the algorithm skips to cell $(i', j)$, where $i'$ is the smallest index $i$ that has an active (i.e., unpruned) neighbor in the previous column $col_{j-1}$. The speedup resulting from pruning is a function of the number of skipped cells.
Figure 4.1: Continuous dynamic programming with pruning. (a) If the pruning classifier outputs 1 the algorithm advances to the next cell in the current column $j$. (b) If the pruning classifier outputs 0 the algorithm skips to the cell with the smallest index $i$ that has an active (unpruned) neighbor from the previous column $j - 1$. The speedup is a function of the number of skipped cells.

The complete description of CDPP is provided in Algorithm 2. The input to the algorithm, once more, is the model $M$, a set of model-dependent pruning classifiers $C_i$, the current input frame $j$, the current input feature vector $Q_j$, and the previous column $col_{j-1}$. The output is the current sparse column vector. In order to maximize efficiency we chose a sparse vector representation that enables fast individual element access, while keeping the number of operations proportional to the density of the DP table (the number of black pixels in Figure 1.5). The sparse vector is represented by a pair $<ind, list>$, where $ind$ is a vector of pointers of size $m$ (the model sequence length), and is used to reference elements of the second variable $list$. The variable $list$ is a singly-linked list, where each list node contains the cumulative distance $D(i,j)$, the index $i$ of the corresponding model frame, and potentially other useful information, such as the warping path length. The length of $list$ corresponds to the number of active matching hypotheses that have not been pruned.

Line 1 of the algorithm (see Algorithm 2) sets the model state to 1, and Line 2 initializes
Algorithm 2: The CDPP algorithm.

input: model M, pruning classifiers C_i, input frame j, input feature vector Q_j,
       and previous sparse column vector <ind_{j-1}, list_{j-1} >.
output: current sparse column vector <ind_j, list_j >.

1. i = 1;
2. ptr = ind_{j-1}(0); // Set pointer to the first node of the previous sparse
   column vector
3. // Loop over model feature vectors
4. while i ≤ m do
5.   // First case: no pruning
6.   if C_i(Q_j) == +1 then
7.     nl = new element; // Create new node
8.     // Update cumulative matching cost
9.     nl → D = min{ind_j(i-1) → D, ind_{j-1}(i-1) → D, ind_{j-1}(i) → D} + d(i, j);
10.    // Append node to list
11.    append(list_j, nl);
12.    ind_j(i) = nl; // Set pointer of current sparse column vector to
13.    // current node
14.    i = i + 1; // Proceed with the next cell
15.   else
16.     // Second case: pruning
17.     // if previous column is empty then break and proceed with the
18.     // next input feature vector
19.     if isempty(list_j-1) then
20.        break;
21.     // if current cell has no neighbors from previous column
22.     if ind_{j-1}(i) == NULL then
23.        // Loop until end of previous list or until previous node
24.        while ptr→next != NULL and ptr→next→i ≤ i do
25.            ptr = ptr→next;
26.        end
27.        // if reached the end of previous column then break
28.        if ptr→next == NULL then
29.           break;
30.        // Skip to next cell which has a neighbor in previous column
31.        i = ptr→next→i;
32.     else
33.        // Otherwise, if current cell has a neighbor from the
34.        // previous column, then proceed with the next cell
35.        i = i + 1;
36.     end
37. end
38.
the pointer to the head of the previous column list. Lines 4-24 are executed for all possible
states \( i \), \( 1 \leq i \leq m \). Line 4 tests the output of the pruning classifier. As mentioned before,
there are two cases corresponding to the output of the classifier. Lines 5-10 correspond
to the case of no pruning. If the output is 1 then a new node is created (Line 5), the
cumulative distance \( D \) and state index \( i \) are stored in the node (Lines 6-7), the node is
added to the list (Line 8), the pointer to the newly created node is assigned to the variable
\( \text{ind} \), and the index \( i \) is incremented by 1. Lines 10-24 correspond to the case of pruning.
If the classifier outputs 0 then if the previous column vector is empty then the matching
algorithm can break and proceed to the next input frame \( j + 1 \) (Lines 12-13). Otherwise, if
the neighboring node from the previous column has been pruned (Line 14) then we advance
in the previous vector until the next node contains an index greater than \( i \) or until the end
of the list has been reached (Lines 15-17). If the end of the list has been reached then the
algorithm breaks (Lines 18-19). Otherwise, \( i \) is set to the index of the next node. If the
neighboring node from the previous column has not been pruned then \( i \) is incremented by
1 (Lines 21-22).

Algorithm 2 is invoked separately for every gesture model \( M^g \). For illustration purposes
we show it for a single model. After the algorithm has been invoked for the current input
frame \( j \) and for all the models, the end-point detection algorithm of Chapter 5 is invoked.

The CDPP algorithm was presented for input which consists of a single feature vector
per frame. The extension to handling multiple feature vectors per frame is analogous to
the extension from DTW to DSTW presented in Section 3.4. In the online version, instead
of maintaining the previous and current sparse column vectors, we have to maintain the
previous and current sparse matrices (slices). The size of each slice is \( m \times K \), where \( m \) is
the number of model states, and \( K \) is the number of candidates.

### 4.3 Learning the Pruning Classifiers

The previous section showed how the pruning classifiers are used to speed up the online
spatiotemporal matching stage. This section describes how those pruning classifiers are
learned from training data off-line. The objective of the pruning classifiers learning methods is to maximize the efficiency, or the amount of pruning, and at the same time to minimize loss in spotting and recognition accuracy. Minimizing loss in accuracy can be obtained by minimizing the chance of pruning the optimal alignments between the gesture model and examples from the same gesture class, which are embedded in a continuous input stream.

We use the observation pruning classifiers of Equation 4.2 to illustrate the classifier learning methods. In the first method we devised, the parameters of those pruning classifiers were learned from training data only [3]. In this case, the objective of the classifier learning method is to learn the threshold \( \tau(i) \) for each classifier \( C_i \). The thresholds \( \tau(i) \) are learned as follows: first, the model is learned using all training examples. Then, the model is aligned, using DTW, with all the training examples of gestures from the same class. For all training examples both spatial and temporal segmentations are available to the system, and therefore DTW can be applied to those examples. The distances between model state \( M_i \) and all feature vectors \( Q_j \) that are matched with \( M_i \) by DTW are saved. The threshold \( \tau(i) \) is set to the maximum distance among those distances.

Setting the thresholds as specified guarantees that, for each training example, the correct warping path according to DTW will not be pruned out, even when the training example is embedded in an arbitrary super-sequence. However, the thresholds thus learned may overfit the training data, and may not generalize well to yet unseen test data. In the worst case, the pruning classifiers will reject a significant fraction of correct warping paths.

To illustrate the worst case, suppose that we have, for a particular gesture model \( M = (M_1, \ldots, M_m) \) (ignoring the dummy state \( M_0 \)), a set \( T \) of \( N \) training examples, and a set \( S \) of \( N \) test examples. Assume that training and test examples are independent identically-distributed (iid), and that, in any sequence, any pair of observations is conditionally independent, given the states that those observations are assigned to by the correct warping path. Let \( S_s \) be a test example, and \( W_s = (i_1, j_1, k_1), \ldots, (i_T, j_T, k_T) \) be the correct warping path between \( S_s \) and the model \( M \). We want to answer the following question: what is the probability that \( W_s \) will be pruned out? The correct warping path
will be pruned out if, for at least one triple \((i, j, k)\), it holds that \(d(i, j, k) > \tau(i)\).

Let \(\tau'(i)\) be the value we would have computed for \(\tau(i)\) if we had used \(S\) as the training set, instead of \(T\). Since both the training and the test examples are iid, there is a 0.5 probability that \(\tau'(i) > \tau(i)\), in which case for at least one test example (the one from which \(\tau'(i)\) was computed), the correct warping path will be pruned out, because it will match a feature vector \(Q_{jk}\) to state \(M_i\) so that \(d(i, j, k) = \tau'(i) > \tau(i)\). Therefore, the probability that a random sequence \(S_s \in S\) would be rejected because of \(\tau(i)\) is at least \(\frac{1}{2N}\). Since we learn one threshold \(\tau(i)\) for each state \(M_i\), we learn a total of \(m\) such thresholds, and the probability that a random test example \(S_s\) will not be rejected by any \(\tau(i)\) is at most \((\frac{2N-1}{2N})^m\).

The above analysis corresponds to a worst-case scenario. However, this problem is also observed in practice. In our experiments a large fraction of correct warping paths are pruned out if we learn thresholds \(\tau(i)\) using the learning method just described. To reduce overfitting and the chance of pruning the optimal paths between the gesture model and test gestures from the same gesture class, leave-one-out cross validation is used to learn a single parameter we call the “expansion factor” \(\epsilon\).

The cross-validation procedure for learning the expansion factor is depicted in Algorithm 3. The inputs to the algorithm are \(L\) (positive) training examples from the same gesture class, and \(L\) (positive) validation examples from the same gesture class. (The number of training examples and validation examples need not be equal, however it simplifies the algorithm notation). The output is the expansion factor \(\epsilon\). Each round \(l\) \((l = 1, \ldots, L)\) of leave-one-out cross validation is carried out as follows (Lines 1-19): the data is split into two disjoint sets: the validation set \(V = \{V^l\}\), which consists of the single left-out validation example for this round (Line 3), and the training set \(T = \{T^1, \ldots, T^L\} - \{T^l\}\), which consists of all training examples excluding the example \(T^l\). (Line 2). First, all the thresholds are set to zero (Lines 4-6) to enable the max operation in Line 14. Then, the model \(M\) is learned from the set of training examples \(T\) (Line 7). The model thresholds \(\tau_l(i)\) are learned as before from the training set (Lines 8-16): every training example \(T^{l'}, l' \neq l\)
(Lines 8-11) is aligned to the model $M$ using DTW (Line 12). All the distances between model state $M_i$ and the training observations that match model state $M_i$ are recorded, and $\tau_l(i)$ is set to the maximum among those distances (Line 14). However, using the thresholds $\tau_l(i)$ may erroneously prune the left-out example $V^l$. The “expansion factor” for this round $l$ is the minimal scalar $\epsilon_l$ (Line 18), such that if we add it to all the thresholds $\tau_l(i)$, then using the resulting thresholds $\tau_l(i) + \epsilon_l$, that left-out example will not be pruned. The final expansion factor $\epsilon$ is set to the maximum among all the $\epsilon_l$.

One can show that, in the worst-case scenario discussed above, adding the expansion factor to each $\tau(i)$ reduces the probability that a random test gesture from the same class will be pruned out to $\frac{1}{N}$ compared to $1 - \left(\frac{2N-1}{2N}\right)^m$ without the expansion factor. In the digit recognition application described in the experiments (see Chapter 6), where $N = 30$ and $m = 10$, the probability that the correct warping path for a random test sequence will be pruned out is reduced to 0.033 compared to 0.154 without the expansion factor.
Algorithm 3: Learning the expansion factor for the observation pruning classifiers

**input**: Training examples \( \{T^1, \ldots, T^L\} \) from the same class.

Validation examples \( \{V^1, \ldots, V^L\} \) from the same class.

**output**: Expansion factor \( \epsilon \).

1. for \( l = 1 : L \) do
   2. \( T = \{T^1, \ldots, T^L\} \setminus \{T^l\} \)/ Training set
   3. \( V = \{V^l\}\)/ Validation set
   4. // Initialize thresholds to zero to enable max operation in Line 14
   5. for \( i = 1 : m \) do
      6. \( \tau_l(i) = 0 \)
   7. end
   8. \( M = \text{learn\_model}(T) \)/ For all training examples
   9. for \( l' = 1 : L \) do
      10. // excluding training example \( T^l \)
      11. if \( l' == l \) then
         12. continue;
      13. end
      14. // Find the optimal path between the model and the current
          training example using DTW
      15. \( W^*_l = \text{DTW}(M, T^{l'}) \)
      16. // For every model feature vector ...
      17. for \( i = 1 : m \) do
         18. // Find the observation feature vector with the maximal
             distance
         19. \( \tau_l(i) = \max_{(j_l,j_l')} \{d(M_i, T^{l'}_{j_l'}), \tau_l(i)\} \)
      20. end
   21. end
   22. // Validation
   23. // Find the optimal path between the model and the validation
       example using DSTW
   24. \( W^*_l = \text{DSTW}(M, V^l) \)/ Find the expansion factor for round \( l \)
   25. \( \epsilon_l = \max_{(i,j_l,k_l)} \{d(M_i, V^l_{j_l,k_l}) - \tau_l(i)\} \)
   26. end
   27. // Set the expansion factor to the maximum among the expansion factors
      of all validation rounds
   28. \( \epsilon = \max\{\epsilon_l\} \)
Chapter 5

Spotting

Pattern spotting is the problem of detecting when a certain pattern of interest occurs in an input stream. Pattern spotting therefore really involves two problems: deciding whether or not the pattern occurs in the input, and estimating when the pattern starts and ends in the input. When one pattern appears as a part of a longer pattern, then whenever the long pattern appears in the input, the system may falsely and prematurely detect the occurrence of the nested pattern. By incorporating reasoning about nested patterns in the spotting algorithm, such false detections may be avoided. This is the main contribution of this chapter.

5.1 Gesture Spotting Algorithm

The matching algorithms described in the previous chapters compute, for a given model and a particular input frame, the matching cost between the model and the input subsequence ending at that frame. The matching costs for all models are computed in a similar way. Those matching costs are fed into the spotting algorithm, which decides if a gesture has been detected.

The proposed gesture spotting algorithm consists of two steps: the first step updates the current list of candidate gesture models. The second step uses a set of rules to decide if a gesture was spotted, i.e., if one of the candidate models truly corresponds to a gesture performed by the user. The end point detection algorithm is invoked once for every input frame $j$. In order to describe the algorithm we need the following definitions:
• **Complete path:** a legal warping path $W(M_{1:m}, Q_{j':j})$ matching an input subsequence $Q_{j':j}$ ending at frame $j$ with the complete model $M_{1:m}$.

• **Partial path:** a legal warping path $W(M_{1:i}, Q_{j':j})$ that matches an input subsequence $Q_{j':j}$ ending at the current frame $j$ with a model prefix $M_{1:i}$.

• **Active path:** any partial path that has not been pruned by CDPP.

• **Active model:** a model $M^g$ that has a complete path ending in frame $j$.

• **Firing model:** an active model $M^g$ with a matching cost below the detection acceptance threshold.

• **Subgesture relationship:** a gesture class $g_1$ is a subgesture of another gesture class $g_2$ if gesture model $M^{g_1}$ matches well a part of gesture model $M^{g_2}$. In this case, $g_2$ is a supergesture of $g_1$.

At the beginning of the spotting algorithm the list of candidates is empty. Then, at every input frame $j$, after all the spatiotemporal matching costs have been updated, the best firing model (if such a model exists) is considered for inclusion in the list of candidates, and existing candidates are considered for removal from the list. The best firing model may be different depending on whether or not subgesture reasoning is carried out, as described below. For every new candidate gesture we record its class, the frame at which it has been detected (or the end frame), the corresponding start frame (which can be computed by backtracking the optimal warping path), and the optimal matching cost. The algorithm for updating the list of candidates is described below. The input to this algorithm is the current list of candidates, the state of the DP tables at the current frame (the active model hypotheses and their corresponding scores), and the lists of supergestures. The output is an updated list of candidates. Steps that involve subgesture reasoning are used in the algorithm CDPP with subgesture reasoning (CDPPS) only, and are marked appropriately.

1. Find all firing models and continue with following steps if the list of firing models is nonempty.
2. CDPPS only: conduct subgesture competitions between all pairs of firing models. If a firing model \( M^{g_1} \) is a supergesture of another firing gesture model \( M^{g_2} \) then remove \( M^{g_2} \) from the list of firing models. After all pairwise competitions the list of firing models will not contain any member which is a supergesture of another member.

3. Find the best firing model, i.e., the model with the best score.

4. For all candidate model \( M^{g_i} \) perform the following four tests:

   (a) CDPPS only: if the best firing model is a supergesture of any candidate \( M^{g_i} \) then mark candidate \( g_i \) for deletion.

   (b) CDPPS only: if the best firing model is a subgesture of any candidate \( M^{g_i} \) then flag the best model to not be included in the list of candidates.

   (c) If the score of the best firing model is better than the score of a candidate \( M^{g_i} \) and the start frame of the best firing model occurred after the end frame of the candidate \( M^{g_i} \) (i.e., the best firing model and candidate \( M^{g_i} \) are non-overlapping, then mark candidate \( M^{g_i} \) for deletion.

   (d) If the score of the best firing model is worse than the score of a candidate \( M^{g_i} \) and the start frame of the best firing model occurred after the end frame of the candidate \( M^{g_i} \) (i.e., the best firing model and candidate \( M^{g_i} \) are non-overlapping, then flag the best firing model to not be included in the list of candidates.

5. Remove all candidate models \( M^{g_i} \) that have been marked for deletion.

6. Add the best firing model to the list of candidates unless it has been flagged to not be included in that list.

After the list of candidates has been updated then if the list of candidates is nonempty then a candidate may be “spotted”, i.e., recognized as a gesture performed by the user if:
1. CDPPS only: all of its active supergesture models started after the candidate’s end frame $j^*$. This includes the trivial case, where the candidate has an empty supergesture list, in which case it is immediately detected.

2. all current active paths started after the candidate’s detected end frame $j^*$.

3. a specified number of frames have elapsed since the candidate was detected. This detection rule is optional and should be used when the system demands a hard real-time constraint. This rule was not used in our experiments.

To the best of our knowledge the idea of explicit reasoning about the subgesture relationship between gestures, as specified in steps 2, 4a, and 4b of the candidates update procedure and step 1 of the end-point detection algorithm, is novel.

Once a candidate has been detected the list of candidates is reset (emptied). In addition, the dynamic programming tables of all models have to be reset. There are two cases to consider. If the reason for spotting was that all the active path hypotheses started after the end frame of the detected gesture, then the DP tables need not be reset and their update can continue from the current frame as usual. Otherwise, all active path hypotheses that started before the end frame of the detected gesture have to be reset. The problem is that the cell hypotheses of those reset active paths have to be recomputed assuming the DP table computations start anew at the frame following the end frame of the detected gesture. There is no procedure that we are aware of that is more efficient than simply resetting all tables and starting anew at the frame following the end frame of the detected gesture. If the system response time is set too high and consequently there is a significant lag between the current frame and the end frame of the detected gesture, this can result in a significant overhead. In practice, the smallest response time should be selected with the highest accuracy.
5.2 Learning Subgesture Relations

The proposed gesture spotting algorithm requires the knowledge about the subgesture relation between gestures. A gesture $g_1$ is a subgesture of another gesture $g_2$ if $g_1$ is similar to a part of $g_2$ under the same similarity model of the matching algorithm. Subgesture relations can be manually specified using domain knowledge. For example, in the digit recognition task the digit “5” can be considered a subgesture of the digit “8” if the features used are 2D positions and if the corners of “5” are not modeled. However, when subgesture relations are less obvious and when the number of gesture models is large, an automated learning method is called for.

One way to learn subgesture relations is to run the spotting algorithm without subgesture relations on example input sequences, and then look for places where the spotter makes a substitution error, i.e., where a gesture is confused as another gesture. The source of those errors may be due to a subgesture relation. Instead of running the spotter on the entire input sequence we can save time by running it only on input segments that correspond to occurrences of gestures. Furthermore, instead of running the spotter on a segmented gesture we can run only the matching algorithm CDP with pruning (CDPP). If for any frame $j$ during the runtime of CDPP, the last state $m$ of the model has been reached then this may be an indication of a subgesture occurrence.

Given a particular gesture model $M$ and an example $Q^g$ from a different gesture model $Q$ we run CDPP on this model-exemplar pair. If there exists a warping path $W = ((1,j_s), \ldots, (m,j_e))$ for some $1 \leq j_s \leq j_e \leq n$ that was not pruned by the algorithm then that means that the gesture model $M$ matches well a subsequence of the exemplar $Q_{j_s:j_e}$, and therefore $M$ is a subgesture of $Q$. We can repeat this process for all the remaining examples $g$ of $Q$, and the number of times that $M$ matches well a subsequence of $Q$ can serve as a confidence measure of $M$ being a subgesture of $Q$.

Algorithm 4 describes the subgesture learning algorithm. The input to the algorithm is all models $M^g$ and all gesture examples $Q^{(g,e)}$, where $e$ is the gesture example index.
The goal of the algorithm is to construct a subgesture table $ST(g_1, g_2)$, such that an entry $ST(g_1, g_2) = 0$ indicates that $g_1$ is not a subgesture of $g_2$, and $ST(g_1, g_2) > 0$ indicates that $g_1$ is a subgesture of $g_2$, where the actual value of $ST(g_1, g_2)$ can serve as a measure of confidence.

The subgesture table learning algorithm is summarized in Algorithm 4. The algorithm first considers all possible pairs of gestures (Lines 1-6). For a given model gesture $M_{g_1}$ and all example gestures $Q^{(g_2,e)}$ from another class (Line 7) we run the CDPP algorithm (Line 9). If $g_1$ is a subgesture of $g_2$ then there exists at least a single warping path, that has not been pruned by the classifiers, and that aligns the entire model $M_{g_1}$ with a subsequence of the example gesture $Q^{(g_2,e)}$. The sparse column vector $<ind_{je}, list_{je}>$ will contain a node, which corresponds to the final state index $m_{g_1}$, and therefore the corresponding entry in the subgesture table will be incremented by one (Line 10). If on the other hand $g_1$ is not a subgesture of $g_2$ then no path exists that aligns the entire model $M_{g_1}$ with any subsequence of the example gesture $Q^{(g_2,e)}$. None of the column vectors returned by the CDPP algorithm will contain a node, which corresponds to the final state index $m_{g_1}$, and therefore the corresponding entry in the subgesture table will not be incremented.

5.3 Gesture Verification

In Section 5.1 we described the spotting rules that decide if and when to spot a particular gesture. The inputs to the spotting algorithm are the matching costs of all gesture models, that were computed by the spatiotemporal matching algorithm. Occasionally, a candidate gesture with a sufficiently low matching cost may fire and be incorrectly classified. This may happen for a number of different reasons: the features used for matching may not be sufficiently informative; the local similarity measure may not be entirely appropriate; the local view that Dynamic Programming has about the shape of the trajectory. (This does not contradict the fact that Dynamic Programming finds the global optimal alignment.) Gesture verification can be used to decide whether to accept or reject the claimed identity of the gesture.
Algorithm 4: Subgesture table learning algorithm.

\textbf{input}: Gesture models $M^g, 1 \leq g \leq G$ and gesture examples $Q^{(g,e)}, 1 \leq e \leq N_g$.
\textbf{output}: Subgesture Table $ST(g_1,g_2)$.

// Consider all pairs of gesture
\begin{algorithmic}
  \For{$g_1 = 1 : G$}
    \For{$g_2 = 1 : G$}
      $ST(g_1,g_2) = 0$; // Initialize gesture table entry
      \If{$g_2 == g_1$}
        continue;
      \EndIf
      // Loop over all examples from another gesture class
      \For{$e = 1 : N_{g_2}$}
        \For{$j = 1 : n_{(g_2,e)}$}
          // Run CDPP using a gesture model and an example from another gesture class
          $<ind_j, list_j> = CDPP(M^{g_1}, C, j, Q^{(g_2,e)}, <ind_{j-1}, list_{j-1}>)$;
          // if there exists an optimal path that matches the input with the entire model
          \If{$ind_j(m_{g_1}) \neq NULL$}
            // then increment the corresponding table entry by one
            $ST(g_1,g_2) = ST(g_1,g_2) + 1$;
            break;
          \EndIf
        \EndFor
      \EndFor
    \EndFor
  \EndFor
\end{algorithmic}

Verification is an integral part of most existing keyword spotting systems in speech [94]. Those systems are typically divided into two components: the first component is a Viterbi decoder that takes as input the speech signal, and makes use of the keyword models, the filler (non-keyword) models, and the language model to output a stream of hypothesized keywords and fillers along with their matching scores. The second component is a verifier that uses a rule to decide if to accept or reject the hypothesized keywords.
5.4 Classical Hypothesis Testing

The verification step is generally considered as a hypothesis testing problem, where the Neyman-Pearson hypothesis formulation is employed [44]. Under the Neyman-Pearson framework there are two hypotheses about the state of nature. In the context of gesture spotting, the null hypothesis $H_0$ represents the case where the input subsequence is the correct key gesture, and the alternative hypothesis $H_1$ corresponds to the case where the input subsequence is a non key gesture. Given some features $O$ corresponding to a possible occurrence of a gesture, the goal of the verifier is to generate a score $S_O$ representing the acceptance degree of confidence for that gesture. A hypothesis test can be formulated by defining a decision rule $\delta$ as follows

$$
\delta(Q) = \begin{cases} 
0, & \text{if } S_O > \tau \quad \text{(accept } H_0) \\
1, & \text{if } S_O \leq \tau \quad \text{(accept } H_1) 
\end{cases}
$$

(5.1)

where $\tau$ is a constant that partitions the space into two regions, namely the acceptance region and the rejection region. Two types of errors can be made when performing a classical hypothesis test:

- A type I error occurs when $H_0$ is rejected but the key gesture is present.
- A type II error occurs when $H_0$ is accepted but the key gesture is not present.

In the context of spotting, a type I error corresponds to a deletion error or a miss, and a type II error corresponds to an insertion error. If multiple gesture models are considered then a type II error may correspond to a substitution (or confusion) error, where a certain key gesture is accepted while another key gesture is present. Gesture spotting performance measures are based on the tradeoff between these two types of error. Those measures will be discussed in the experiments chapter.
5.5 Verification Classifier

The question is which features $O$ to use and how to design a scoring function $S_O$ that yields the best error tradeoff. The features $O$ may include the matching cost of the spotted gesture, the number of competing active gesture candidates, the matching costs of those competing gesture models, etc. The most widely used scoring functions $S_O$ in gesture spotting applications are based on some form of likelihood ratio between the gesture model and the non-gesture model [67]. In this section, we show how discriminative classifiers based on trajectory features and appearance features can be used for verifying hypothesized gestures.

Trajectory-Based Classifier

One set of features that could be useful for verification are based on the trajectory of the input subsequence, which matches the hypothesized gesture model. The reason such features could be useful is that in contrast to the matching algorithm that has a local view of features and similarity measures, the verification algorithm has access to the entire trajectory, and can therefore use global trajectory features and similarity measures, as well as normalizing geometric transformations, in deciding whether to accept the gesture candidate or to reject it from further consideration.

In contrast to earlier work in gesture spotting we base our gesture verification stage on a discriminative classifier. Let $M^g$ be one of the candidate gesture models that is firing at frame $j_e$. Let $Q^g = (Q^g_{(j_s,k_{j_s})}, \ldots , Q^g_{(j_e,k_{j_e})})$ be the recovered trajectory of the input subsequence that matches model $M^g$, with hypothetical start and end frames, $j_s$ and $j_e$ respectively. We want a verification classifier $C^g(Q^g)$ such that

$$C^g(Q^g) = \begin{cases} 1 & \text{if the claimed identity of } Q^g \text{ is correct} \\ 0 & \text{otherwise} \end{cases}, \quad (5.2)$$

Following our previous work on classifying trajectories [4] the verification classifier can
be a linear combination of classifiers [97], where each individual classifier \( c_j \) corresponds to a subset of features from the set of trajectory features \( Q^g \). The form of the classifier is:

\[
C(Q^g) = \begin{cases} 
1 & \sum_{j=1}^J \alpha_j c_j(Q^g) > 0 \\
0 & \text{otherwise}
\end{cases}
\]

(5.3)

where \( \alpha_j > 0 \) are positive weights. A classifier \( C^g \) is trained for each gesture model \( g \). The goal of the training algorithm is to learn the (weak) classifiers \( c_j \) and the weights \( \alpha_j \). In order to train the classifiers two sets of positive and negative examples have to be provided. The positive examples are simply examples from that gesture class. The question of how to select negative examples is a little harder. One alternative is to select the examples that the spotter incorrectly classified. Those negative examples correspond to the substitution and insertion errors made by the spotter for that model. Those examples are in a sense the hardest negative examples. There are two problems with such examples. First, it may be possible that the input subsequences that caused insertions appear visually similar to the gesture model under consideration, and in fact they should not be really considered as negative examples. In case of substitutions two models can give similar scores because the similarity model is too weak. The second problem is that the number of such negative examples may be too small for training.

Another alternative is to select examples from other classes as the negative examples for the given class. Here again, there are a few alternatives, but two are most common [2]. In a one-vs-all classifier the negative examples are all the examples from other classes. In a pairwise classifier the negative examples are taken from a single different class. The hypothesized gestures in a given input frame \( j \) can be verified by combining the outputs of the corresponding classifiers.

**Appearance-Based Classifier**

An additional verification strategy is to use features which provide complementary information to that provided by the features used in the matching process. In the sign spotting...
experiment reported in Chapter 6, we use trajectory features, i.e., location and motion, in the matching process. However, as mentioned in Chapter 2, the basic components of a sign gesture consist of hand shape and hand orientation, in addition to location and motion [106]. Hand shape and orientation are encoded in the hand’s appearance in the image.

Hand appearance features can be used in different stages of the vision process. For instance, hand appearance can be used in the early detection stage. Hand detection methods based on boosted classifiers [63, 81] became popular following the success of the Viola-Jones face detector [112], which is both accurate and efficient. However, the human hand is an articulated object with many degrees of freedom, and has significantly higher variation in appearance compared to a face. As a direct consequence, a large number of hand images are required for training the classifiers; furthermore, as the number of hand appearance classifiers increases the online hand detection algorithm may become too slow for practical applications.

Another alternative is to use hand appearance features in the gesture matching stage. The main problem of using hand appearance in the matching stage is that the input hand image has to be registered very well with the hand appearance models or templates. Normalization of hand images is a challenging task, due to in part the lack of stable landmarks that can be reliably matched. Most similarity measures such as normalized correlation, chamfer distance, shape context distance, and classifiers based on Histogram of Gradients will give meaningless results when registration is poor [108].

We propose to adopt a cascade framework, where simple and efficient features such as location and motion are used for matching and then the more informative but more computationally demanding appearance features are used only for verification of a hypothesized gesture. After the matching stage the 2D hand trajectory can be recovered from the optimal warping path, and then hand appearance features can be extracted at every trajectory location, and evaluated by the verifier.

The key for a useful appearance-based verification is selecting points in the gesture where the hand has a stable and reliable appearance. Every gesture may have a different
number of distinct appearances which can be used for reliable verification. For example, in sign language most signs are defined by distinct start and end hand shapes, and different parts of the sign may have stable configurations. If the start and end shape are the same then the shape will be the same throughout the gesture. If the start and end shape are different then there will be a gradual transition between those shapes. However, stable hand shapes do not always translate to stable hand appearances. Hand appearance also depends on the hand orientation with respect to the camera, which may change throughout the gesture.

Assuming we have identified key points in the gesture where hand appearance is stable, we can learn an appearance-based classifier for each of those key points. The hand appearance feature that we use is the Histogram of Gradients or HOG feature [36]. The HOG representation has several advantages. It captures edge or gradient structure that is very characteristic of hands, and it does so in a local representation with some degree of invariance to local geometric and photometric transformations.

The verification classifier we use is the linear Support Vector Machine (SVM) [29]:

\[
C^g(x) = \begin{cases} 
1 & w_1^T x + w_0 > 0 \\
0 & \text{otherwise}
\end{cases},
\]

where \(x\) is the HOG feature vector extracted from the hand image patch, and \(w_0\) and \(w_1\) are the parameters of the linear SVM. The SVM classifier parameters are learned from several hundred positive training examples of the target hand appearance, and several thousands negative examples of indoor and outdoor image patches containing no hands.

In our experiments, we applied verification to a sign that has the same hand appearance for the entire duration of the gesture. We thus avoided the problem of deciding which appearance-based classifier to apply to which hand image in the sequence. In this case, given a hypothesized sign of length \(n\) represented by a sequence of HOG features \(x_1, \ldots, x_n\), we apply the appearance classifier to each feature \(x_i\), and normalize the classifier scores
(votes) by the sequence length:

\[ v = \frac{\sum_{i=1}^{n} C^n g(x_i)}{n}. \]  \hspace{1cm} (5.5)

We then compare the normalized voting score \( v \) to a threshold, in order to decide whether to accept or reject the hypothesized sign.
Chapter 6

Experiments

In order to evaluate the performance of our gesture spotting and recognition system we have collected two real-world video datasets: a digit gesture dataset and an American Sign Language (ASL) dataset. The digit gesture video clips depict users gesturing the ten digits in front of a camera, and the goal of the system is to recognize the digits as they are gestured. The test dataset consists of 300 digit examples gestured in continuous video. The ASL video clips depict ASL utterances signed by native ASL signers, and the goal of the system is to spot the occurrences of individual signs. The ASL test dataset consists of 240 sign examples of 24 distinct signs.

Two types of experiments were conducted with the dataset of digit gestures. One type of experiment is isolated gesture recognition, where the temporal extent of a test gesture is known, and the test video clip start and end frames correspond to the gesture start and end respectively. The other type of experiment, which reflects a more realistic application, is continuous gesture recognition, where the temporal extent of test gestures is unknown, and the test video consists of a sequence of interleaving gestures and non-gesture motion patterns. The experiment conducted with the ASL dataset is continuous sign spotting, where the temporal extent of test signs is unknown, and the test video clips are of ASL utterances, which consist of sequences of signs.

The main difference between the test sets collected for our experiments compared to those used in other works is that in other works the users often wear long sleeved shirts [20, 77, 101, 124], they gesture in front of a static uniform background [20, 77, 101, 124], and the cameras are often placed in such a way to avoid hand over face occlusions [101, 102].
In contrast, in our experiments the users and signers can wear short sleeved shirts, the background may be arbitrary (e.g., an office environment in the digit gesture dataset) and even contain other moving objects, and hand over face occlusions are allowed. Most gesture spotting and recognition algorithms will fail under such conditions [101, 75, 67], because they assume that the trajectory of the hand can be reliably extracted using a hand tracker, however most hand trackers will fail in such ambiguous situations. Furthermore, most hand trackers have to be initialized in the first frame of a sequence [51, 122, 107]. In our system such initialization is not required. In the absence of a natural competing gesture recognition system to compare with, we compare the performance of our system when it uses multiple candidate hand location hypotheses to its performance when it uses only the best hand location hypothesis. The objective of the experiments described in this chapter is to measure the performance of the gesture spotting and recognition system in terms of accuracy and speed.

The chapter begins with presenting the performance measures used in evaluating the proposed system’s performance. Then, details are given about the computer setup and software used to implement the system, the image capture setup used to collect the video data, and certain issues related to the algorithm implementation. Finally, the digit recognition experiments and the sign spotting experiments are then described, and for each experiment, quantitative experimental results are provided and discussed.

### 6.1 Performance Measures

The proposed system is quantitatively evaluated in terms of accuracy and computation time. In order to measure accuracy, every gesture appearing in the input video clips was manually labeled with three pieces of information: the gesture class, the gesture start frame and the gesture end frame. The start and end frames of the digit gestures were manually labeled using a standard video file viewer. The start and end frames of the ASL signs were annotated by linguists using the SignStream software [78]. This information constitutes the ground truth to which the system’s estimates are compared. The accuracy
of the system is computed by comparing the system’s estimates to the ground truth labels. We measure computation time both in seconds and in the number of operations required by the algorithms.

**Accuracy**

To measure the system’s accuracy for the isolated digit recognition system, we use classification accuracy, which is the ratio of correctly recognized gestures to the total number of gestures. Classification error is simply the complement of classification accuracy, namely, the ratio of incorrectly recognized gestures to the total number of gestures.

To measure the system’s accuracy for the continuous digit recognition system, we can use the gesture error rate (named after the word error rate (WER) in speech [26]), which is analogous to classification error for the isolated gesture recognition. In continuous gesture recognition there are three types of errors: insertion errors, deletion errors, and substitution errors. An insertion error occurs when the spotter reports a nonexistent gesture, a deletion error (or miss) occurs when the spotter fails to detect a gesture, and a substitution error (which is a combination of an insertion and a miss) occurs when the spotter falsely classifies a gesture as another gesture. If there is a total of $N$ test gestures, $I$ insertion errors, $D$ deletion errors, and $S$ substitution errors, then the gesture error rate can be computed as:

$$GER = \frac{S + I + D}{N} \times 100.$$  \hspace{1cm} (6.1)

If the number of correct detections is $C$ then the correct detection rate is simply:

$$P_d = \frac{C}{N} \times 100.$$  \hspace{1cm} (6.2)

We note that for isolated gesture recognition the sum of classification accuracy and classification error is one. In contrast, for continuous gesture recognition the sum of correct detection rate and the gesture error rate can be greater than one.
For the continuous recognition experiments the correct detection rate is always reported. The error that is reported is either the number of substitution errors and the number of insertion errors, or their sum, which is the number of false alarms.

There is no universal definition of when a gesture is considered successfully detected. Some researchers define a correct detection when the estimated gesture end point is within a number of frames from the ground truth gesture end point [75]. Other researchers define a correct detection when the midpoint of the estimated gesture time interval falls within the ground truth time interval [71], or when 50% of the duration of the estimated gesture overlaps with the ground truth gesture and vice versa [118]. We use the latter definition.

One application we have in mind for the ASL sign retrieval experiment is semi-automatic annotation of ASL. In manual annotation the transcriber has to review all video frames and mark the start and end of a particular sign of interest. In the semi-automatic annotation system the goal is, given a sign query, to minimize the number of frames retrieved by the system and that the transcriber has to review. We then report as the accuracy measure the fraction of frames that the transcriber has to review using the system over the total number of video frames (that the transcriber would have to review without the system).

**Computation Time**

Another measure of performance that is used in the evaluation of the gesture spotting system is the average computation time required for a test video sequence. The overall spotting and recognition process is broken down into two stages: the low-level image processing stage which consists of multiple candidate hand detection and feature extraction, and the high-level spotting and recognition stage which consists of spatiotemporal matching and spotting. Reducing the computational time of the low-level stage is important for a real-time application. Even though the spatiotemporal matching algorithm enables more simple and efficient low-level processing, this efficiency is not the main focus of the experiments, and there is still room for improvement. Reducing the search time of the high-level stage is the main focus of the experiments, and so is the tradeoff between effi-
ciency (computational time) and accuracy. The computational time is measured both as the average processing time per frame, and the average number of operations per frame, where the number of operations is simply the number of times any dynamic programming cell has been accessed.

6.2 Implementation Details

Computer Setup and Software

All experiments were conducted on a 2GHz Opteron-based PC with 2GB of RAM running Microsoft Windows XP Professional Operating System.

The source code for learning the gesture models was developed on Matlab Version 7.0.1. The source code for online spotting and recognition was developed on Visual Studio C++ .NET 2003 Version 7.1. We used Intel’s open source computer vision library (OpenCV) to perform basic image processing and some linear algebra. We also used OpenCV’s HighGUI library to capture frames from live video and movie files, and OpenCV window management and display routines to display the image sequences and the graphical annotations. We used the Standard Template Library (STL)’s vector data type and various STL algorithms. We also modified a template circular buffer data type to implement the sliding window concept described in Section 2.1. Finally, we wrote our own sparse matrix class optimized for fast matrix element access.

Hand Detection and Feature Extraction

For both the digit recognition and ASL retrieval experiments hand detection and feature extraction were carried out in a similar fashion. The spatiotemporal matching algorithm has been designed to accommodate multiple hypotheses for the hand location in every frame. Therefore, we can afford to use a relatively simple and efficient preprocessing step for hand detection. Following other researchers [59, 98] we combine color (skin likelihood) and motion (frame differencing); both requiring only a few operations per pixel.
The skin detector computes for every image pixel a skin probability term. There has been significant work on finding the appropriate color models for skin detection and tracking \[60, 128, 100, 53\]. In our experiments, we found that it is sufficient to use generic skin and non-skin RGB color histograms that were learned from many training examples \[53\]. The probability of a skin pixel can be computed using Bayes rule:

\[
P(skin|rgb) = \frac{P(rgb|skin)}{P(rgb|skin) + P(rgb|not\ skin)},
\]

where we assume equal priors \(P(skin) = P(not\ skin) = 0.5\).

The motion detector computes for every image pixel a motion score based on simple frame differencing. The motion score for each pixel is simply the absolute difference of the gray level values between the current and previous frame.

\[
\Delta I_{abs}(x, y) = |I_j(x, y) - I_{j-1}(x, y)|
\]

The skin probability image and the absolute frame differencing image are then multiplied together. The resulting image is then convolved with a box filter the size of a hand to obtain the hand likelihood image. This convolution operation is speeded up using the integral image \[112\]. We then extract from the resulting hand likelihood image the \(K\) subwindows with the highest score. This can be done in several ways depending on the desired amount of overlap between different detection windows. We want to allow some overlap between the windows. Therefore, after finding the first best window, we zero out all pixels that belong to that window in the hand likelihood image. We then convolve the resulting image with a box filter to obtain a new hand likelihood image, and then look for the next subwindow with the highest score in that image. Our hand detection method for \(K = 1\) is shown in Figure 6-1. For all the experiments we used a hand window size of \(40 \times 30\) pixels.

A distinguishing feature of our hand detection algorithm compared to most existing methods (e.g., \[24, 101\]) is that we do not use connected component analysis to find
the single largest component, and associate it with the gesturing hand. The connected component algorithm may group the hand with the arm if the user is wearing a shirt with short sleeves, or with the face, or with any other hand-like objects with which the hand may overlap. In contrast, our hand detection algorithm maintains multiple subwindows for every frame of the video sequence, and some of these subwindows may occupy different parts of the same connected component. The gesturing hand is typically covered by one or more of these subwindows.

For every frame $j$ of the query sequence, $K$ candidate hand regions are found. For every candidate $k$ in frame $j$ a feature vector $Q_{jk}$ is extracted. For both the digit recognition experiment and the sign spotting experiment $Q_{jk} = (x_{jk}, y_{jk}, u_{jk}, v_{jk})$ is a 4D vector. The 2D position $(x, y)$ is the center of window $k$, and the 2D velocity $(u, v)$ is the optical flow averaged over that window. In our implementation, optical flow is computed using a block-based matching method [127].
Example-Based vs. Model-Based Matching

There are two approaches for matching sequences. In an example-based approach a new test input $Q$ is matched with examples $M$ from the database, and the class label of the database example with the smallest distance is assigned to the input. The input can be matched with all database examples. However, this can be computationally expensive, and therefore more often the input is matched with only a few prototypical examples from each class. In a model-based approach each class is represented by one or a few prototypical models or classifiers and the input is assigned the label of the best matching model.

In the first three experiments we used an example-based approach because the lack of sufficient training data to train a model. In those experiments, $M$ is an example time series and every $M_i$ is a feature vector. In this case, we define the local cost measure $d(i,j,k) \equiv d(M_i, Q_{jk})$ between model state $M_i$ and feature vector $Q_{jk}$, to be the Euclidean distance $d(i,j,k) = |Q_{jk} - M_i|$. In the rest of the experiments we used a model based approach, where $M$ is a model and each $M_i$ is a state with an associated observation density. In our experiments, each $M_i$ is associated with a Gaussian observation density with mean $\mu_i$ and covariance $\Sigma_i$ that assigns a likelihood to each observation vector $Q_{jk}$. In this case, we define the local cost measure $d(i,j,k) \equiv d(M_i, Q_{jk})$ between model state $M_i$ and feature vector $Q_{jk}$, to be the Mahalanobis distance $d(i,j,k) = (Q_{jk} - \mu_i)'\Sigma^{-1}(Q_{jk} - \mu_i)$. Therefore, one difference between example-based matching and model-based matching is in the definition of the local cost measure.

Another difference between the model-based approach and the example-based approach is that in the model-based approach the model parameters (in our implementation the states means and covariances) can be learned from training data. To learn those parameters for a particular model we first select a single prototype example based on a minmax criterion: we compute for each example its DTW distances to all other examples and record the maximum among those distances. We then select the example with the minimum among those maximal distances. After a prototype has been selected all other examples are aligned to it using DTW. Then, all the observations from other examples that aligned with a
particular state $M_i$ are used to learn its parameters.

**Appearance-Based Verification**

Verification is an important part of a spotting method, since it may reduce the number of false matches resulting from weak features and similarity measures employed in the matching algorithm. In the ASL Sign retrieval experiment we applied the appearance-based classifier described in Chapter 5 for verifying the identity of the spotted sign.

The goal of the classifier is to decide, for a given image patch, whether or not a particular hand appearance is present in the image. We therefore train an appearance-based binary classifier with a set of images of a particular hand appearance (the positive examples) and a set of images extracted from real background images (the negative examples). The hand images are synthetically generated using the 3D modeling graphics software Poser [87]. The target hand shape is perturbed with varying finger angles, view angles, aspect ratios and illumination. An example hand image is shown in Figure 6:2(a).

From each training example (positive or negative) the HOG features are extracted as described in Reference [36]. The hand window is of size 48 by 48 pixels, which is divided into 64 cells of size 6 by 6. Nine edge orientation bins are evenly spaced between 0 to 180 degrees (unsigned gradient). Edges are detected by the Sobel edge detector, and each pixel votes for its orientation bin by edge magnitude. Bins in each cell are normalized with surrounding 3 by 3 cells using the 2-norm with the clipping technique used in [36]. Neighboring cells overlap by half. The total feature length is 2025.

The training has two stages, where the first step is used to bootstrap the second stage [36]. First the classifier is trained with positive hand images and random selected background patches. Then the classifier collects false positive patches from the background set as tough negative examples. These are combined with original negative training set to train the classifier again. We use a linear SVM trained with LIBSVM [23], slightly modified to allow a large number of dense feature vectors for training. For training of both levels of the classifiers, the number of positive samples is 800 and that of negative samples is 5000.
Figure 6-2: Example images of the ASL “Y” hand shape: (a) A training image which was generated by Poser. (b) An example test image extracted from a sign sequence to be verified. The image patch of size 80 by 80 is extracted from the original image using the trajectory recovered for the hypothesized sign. The image patch is then scanned with a window size of 48 by 48, and the classifier tests each window location for the appearance of a hand. If the classifier score is positive for any of the locations then the “patch-based” classifier would output 1. Otherwise, it would output 0. This process is repeated for all image patches belonging to the hypothesized sign, and then all the patch-based classifiers vote for the verification of the hypothesized sign.

for the first level and 8000 for the second level.

Given the trajectory of the hypothesized gesture, we extract for every trajectory point an image patch with a sufficiently large size to contain the hand. In our implementation, we use an image patch size of 80 by 80 pixels. An example hand image is shown in Figure 6-2(b). The image patch is scanned with a window size of 48 by 48, and the classifier computes a score for every possible location of the window. If one of the classifier scores is positive then we conclude that the particular hand shape is present in that image. We then count the number of times the hand shape is present in the entire (hypothesized) gesture sequence and normalize this count by the sequence length. The normalized score is compared to a threshold in order to decide whether to accept or reject the hypothesized gesture.
Description of Datasets

The video clips for the digits Dataset I were captured, in the early data collection, with a Logitech 4000 Pro color camera at 30 frames per second using an image size of $240 \times 320$. Due to significant number of frame drops, video clips for the next digits Dataset II were captured with a faster Unibrain Firewire color camera at the same frame rate and with the same image size. Both datasets were captured in an office environment with regular office illumination. All sequences were captured using a frontal view of the user’s upper body. Three subjects appear in Dataset I, while ten subjects appear in Dataset II. The start and end frames of the digit gestures were manually labeled using a standard video file viewer.

The video clips for the ASL dataset were captured with a four camera setup and using specialized hardware to accommodate the high bandwidth. However, we only used a single upper body view camera that captured the video at 60 frames per second with an original image size of $480 \times 640$ pixels, which was down sampled to $240 \times 320$ pixels for the experiments. The ASL dataset was captured in a lab environment with controlled illumination. Sequences of one subject were used for spotting signs in ASL utterances, and sequences of another subject were used for spotting signs in ASL stories. The start and end frames of the ASL signs were labeled by linguists using the SignStream software [78].

6.3 Digit Recognition Experiments

For the first set of experiments we implemented a hand-signed digit recognition system. Two datasets were collected. The smaller preliminary Dataset I was collected mainly for the purpose of comparing the performance of DSTW to the performance of DTW on the task of isolated digit recognition. Dataset I is used in Experiments 1,2 and 3. The larger Dataset II was collected mainly for the purpose of evaluating the performance of the entire system on the task of continuous digit recognition.
Isolated Digit Recognition Experiment 1: DSTW vs. DTW

Dataset I consists of video clips of three users gesturing the ten digits in the style of Palm’s Graffiti Alphabet [84] (Figure 6-3). In those video clips the user’s upper body view is
Query and model trajectories

Figure 6-5: Example query trajectory (left) and corresponding model trajectory (right) for a correct match between two users signing the digit 9.

visible, and the user is sitting in front of a cluttered office background. Figure 6-4 shows example images along with digit trajectories exemplars. A total of 270 digit exemplars were extracted from three different types of video clips depending on what the user wore:

- Colored Gloves: for every user 3 examples per digit were stored in the database (See Figure 6-4).
- Long Sleeves: for every user 3 examples per digit were used as queries.
- Short Sleeves: for every user 3 examples per digit were used as queries.

Given a query frame, $K$ candidate hand regions of size $40 \times 30$ were detected as described in Section 6.2. For every candidate hand region $k$ in every query frame $j$, a 4D feature vector $Q_{jk} = (x_{jk}, y_{jk}, u_{jk}, v_{jk})$ was extracted and normalized to a unit hypercube. The query digit was then matched with the model exemplars in the database: for the user-dependent experiments, 30 query digits of one user were matched with 30 database digits of the same user; for the user-independent experiments, 30 query digits of one user were matched with all 60 database digits of the two other users. The class of the query was estimated using the one nearest neighbor (1-NN) rule, and classification accuracy rates were averaged over the three users. Examples of a correct match and a false match are shown in Figures 6-5 and 6-6 respectively.

The purpose of the first experiment is to demonstrate that the DSTW algorithm outperforms the simple DTW algorithm when using a hand detection method based on color and motion [24]. The classification rates depicted in Table 6.1 show a significant (11.1%–21.1%)
Query and model trajectories

Figure 6.6: Example confusion between query digit 3 (left) and model digit 7 (right). In the final segment of the query digit 3 the elbow rather than the hand is falsely matched with the hand of model digit 7.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>User-dep.</th>
<th>User-indep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>DTW</td>
<td>DSTW</td>
</tr>
<tr>
<td>Long Sleeves</td>
<td>81.1</td>
<td>96.7</td>
</tr>
<tr>
<td></td>
<td>76.7</td>
<td>91.1</td>
</tr>
<tr>
<td>Short Sleeves</td>
<td>82.2</td>
<td>93.3</td>
</tr>
<tr>
<td></td>
<td>70.0</td>
<td>91.1</td>
</tr>
</tbody>
</table>

Table 6.1: Classification accuracy results. The results for DSTW are for $K = 8$.

increase in classification accuracy between the simple DTW algorithm, which can only handle a single (best) candidate, and the proposed DSTW algorithm, which can handle multiple candidates. In addition, the graphs in Figure 6.7 show the initial decreasing trend of the classification error rate as $K$ increases. At some point the error rate stops decreasing since additional candidates cause more false matches. The optimal value for $K$ can be estimated for a particular scene using cross validation.

The results in Table 6.1 also show that the classification accuracy rates for the short sleeves sequences are slightly worse than the classification accuracy for the long sleeves sequences. This is to be expected, because the gesturing hand is more accurately localized when the user wears a long sleeved shirt. However, it is important to note that the classification accuracy for the short sleeves sequences would be much worse without handling multiple candidate observations, unless much more sophisticated hand segmentation and detection algorithms were employed.
Isolated Digit Recognition Experiment 2: Translation Invariance

The purpose of the second experiment is to demonstrate the additional benefit of incorporating translation invariance in the DSTW framework as proposed in Section 3.5. In principle, translation invariance could be obtained using translation invariant features such as relative position with respect to the hand position in the first frame, or velocity. However, using relative position is not possible because the hand position in the first frame is not known, and using only velocity as a translation invariant feature causes dramatic drops in classification accuracy. For example, accuracy drops from 85.6% to 22.2% in the user-independent experiment with short sleeves sequences using $K = 12$.

On the other hand, if both absolute position and velocity are included in the feature vector and translation invariance is not handled, then if there is a shift of the gesture in the image plane, the classification error rate will increase. The graph in Figure 6.8 shows the increase of the error rate as a function of (a synthetic) increase in translation, for the user-independent experiment with short sleeves sequences. Clearly, for the translation invariant formulation, the error rate does not change as translation increases, as indicated by the horizontal line in the graph. We also note that the reason that the error rates when using absolute position and velocity are relatively low for small translation is that all gestures were performed approximately at the same location in the image plane.
Continuous Digit Recognition Experiment 3: Single Hand Candidate

In this experiment we compare our proposed gesture spotting algorithms to a baseline algorithm for the case where only a single hand is detected in every frame. The baseline algorithm is Continuous Dynamic Programming (Section 3.3) with a typical set of gesture spotting rules. In particular, we used a global acceptance threshold for detecting candidate gestures, and we used the gesture candidate overlap reasoning described in Section 5.1. The proposed CDP with pruning algorithm (CDPP) is implemented as described in Section 4.2, with the same gesture spotting rules used in the baseline algorithm. The second proposed algorithm, CDPP with subgesture reasoning (CDPPS) includes the additional steps marked in Section 5.1.

To compare the performance of the different algorithms we used video clips of two of the users from Dataset I gesturing the ten digits 0-9 in sequence. For each user we used two types of sequences: three colored glove sequences and three long sleeves sequences. The model digit exemplars were extracted from the colored glove sequences as for the isolated digit recognition experiments, and the long sleeves sequences were used as test sequences. The range of the test sequence lengths is $[1149, 1699]$ frames. The range of the digit sequence lengths is $[31, 90]$ frames. The range of the (in between digits) non-gestures sequence lengths is $[45, 83]$ frames.

For every frame we computed the 2D hand centroid locations and the angle between
the \( x \) axis and the vector connecting two consecutive hand locations. The feature vectors \((M_i, Q_j)\) used to compute the local distance \(d(i, j)\) are the 2D positions only. The classifier used for pruning was combination of two classifiers: one based on the 2D positions and the other based on the angle feature. Those classifiers were trained on the model digits in the offline step. To avoid overpruning we added an \textit{expansion factor} of 20 pixels to the thresholds of all position classifiers and an \textit{expansion factor} of 25 degrees to all angle classifiers. Those expansion factors were manually set to reasonable values. For the spotting algorithm we specified Subgesture Table 6.2, which captures the subgesture relations between digits.

The experimental results are summarized in Table 6.3. For the baseline CDP algorithm we obtained 47 correct detections and 13 false matches. For the proposed CDPP algorithm without subgesture reasoning we obtained 51 correct detections and 9 false matches, and finally for the proposed CDPP algorithm with subgesture reasoning we obtained 58 correct detections and 2 false matches. The two false matches resulted from two examples of the digit 0 that were confused as 6. Compared to CDPP without subgesture reasoning, the proposed CDPP with subgesture reasoning corrected a single instance of the digit “3” initially confused as its corresponding subdigit “7”, four instances of the digit “8” initially confused as its corresponding subdigit “5”, and two instances of the digit “9” initially confused as its corresponding subdigit “1”.

In our experiments CDPP executed 14 times faster compared to CDP in terms of CPU time, assuming feature extraction.

<table>
<thead>
<tr>
<th>Subgesture</th>
<th>Supergestures</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0”</td>
<td>{“9”}</td>
</tr>
<tr>
<td>“1”</td>
<td>{“4”, “7”, “9”}</td>
</tr>
<tr>
<td>“4”</td>
<td>{“2”, “5”, “6”, “8”, “9”}</td>
</tr>
<tr>
<td>“5”</td>
<td>{“8”}</td>
</tr>
<tr>
<td>“7”</td>
<td>{“2”, “3”, “9”}</td>
</tr>
</tbody>
</table>

\( \textbf{Table 6.2:} \) Subgesture table specified as input to the spotting algorithm.
<table>
<thead>
<tr>
<th>Method</th>
<th>CDP Detection Rate</th>
<th>CDPP Detection Rate</th>
<th>CDPPS Detection Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>78.3%</td>
<td>85.0%</td>
<td>96.7%</td>
</tr>
<tr>
<td>False Matches</td>
<td>13</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 6.3:** Comparison of gesture spotting accuracy results between the baseline and the proposed gesture spotting algorithms. The accuracy results are given in terms of correct detection rates and false matches. The total number of gestures is 60.

**Continuous Digit Recognition Experiment 4: Multiple Hand Candidates**

This experiment is intended to evaluate the performance of the entire gesture recognition system that combines spatial and temporal segmentation. For this experiment we used the larger Dataset II which was collected in a similar manner to Dataset I. Dataset II consists of ten users, each user appearing in six sequences: three colored glove sequences, and three short sleeves sequences. The 30 glove color sequences were used for learning the digit models. The 30 short sleeves sequences were used for testing. Therefore, the test set included a total of 300 digit occurrences. Experiments were performed in a user-independent fashion: for each user, that user’s training sequences were *not* used in building the models with which that user’s test sequences were matched.

Given a test frame, $K$ candidate hand regions of size $40 \times 30$ were detected as described in Section 6.2. For every candidate hand region in every test frame, a feature vector was extracted as described in Section 6.2. The 10 DP tables were updated as described in Section 3.4. Single observation classifiers (Section 4.1 Equation 4.2) were used for pruning and expansion factors were learned as described in Section 4.3. The resulting matching costs were fed into the spotting algorithm, which decided whether or not a gesture has ended at that time.

The spotting algorithm used the Subgesture Table 6.4 that was learned from training data. By comparing the learned table to the one specified in the previous experiment it is evident that the learning algorithm discovered meaningful subgesture relations automatically, and in fact the set of learned relations is a proper subset of the manually specified relations.
Table 6.4: Subgesture table learned from training data.

<table>
<thead>
<tr>
<th>Subgesture</th>
<th>Supergestures</th>
</tr>
</thead>
<tbody>
<tr>
<td>“1”</td>
<td>{“0”}</td>
</tr>
<tr>
<td>“4”</td>
<td>{“5”}</td>
</tr>
<tr>
<td>“5”</td>
<td>{“8”}</td>
</tr>
<tr>
<td>“7”</td>
<td>{“2”, “3”}</td>
</tr>
</tbody>
</table>

Given those input parameters to the spotting algorithm, we conducted two experiments, where models are represented by a different number of states. In the first experiment the number of model states was 10 for all models. For the optimal setting of number of candidates ($K = 3$) the correct detection rate obtained was $257/300 = 85.67\%$ and the number of false positives was 54. A speed-up factor of two was obtained using our pruning method, and the processing time for spatiotemporal matching and spotting was 0.5 ms per frame. We note that using the pruning method without learning expansion factors, the correct detection rate was $212/300 = 70.67\%$ and the number of false positives was 37.

In the second experiment the number of model states was selected in proportion to the trajectory length of the digit prototype. Those learned digit models are shown in Figure 6.9. For the optimal setting of number of candidates ($K = 3$) the correct detection rate obtained was $212/300 = 70.7\%$ without subgesture reasoning and the number of false alarms was 72, where all false alarms were due to substitution errors, most of which were caused by early detection of subgestures. Subgesture reasoning corrected 59 of the 72 substitution errors, leading to an overall detection rate of $271/300 = 90.3\%$. Table 6.5 shows the number of correct detections and the number of substitution errors per digit, with and without subgesture reasoning.

The speed-up obtained by pruning in our experiment is a somewhat pessimistic estimate. Pruning depends on several factors including:

1. The similarity between gesture prefixes. In our experiment all digits start at the top, and therefore it is difficult to prune early on.

2. The number of model states. The fewer the number of states the less chance there is
to benefit from pruning. In our experiment we use ten states per digit model.

3. The ratio between gesture time and total time. A small ratio means that the periods of in between gestures are relatively long, and therefore there is more chance of pruning during that time (since those in-between periods would tend to match badly with the models). To verify the influence of the ratio of gesturing time over total sequence time, we conducted an experiment where we varied the periods of rest between gestures. Figure 6.10 shows that the amount of pruning increases as the rest time in between gestures increases.

For all the results reported for the digit experiments we used the single observation classifier (Section 4.1 Equation 4.2). In other experiments we considered two transition classifiers (Section 4.1 Equations 4.3 and 4.4). The first classifier we considered is based on the cosine of angle between the direction of the vector between two successive detections and the estimated flow direction computed for the first detection. If optical flow estimates were accurate enough those two directions would be consistent and the cosine would be
### Table 6.5: Classification accuracy results for continuous digit recognition.

<table>
<thead>
<tr>
<th>Digit</th>
<th># Correct Detections w/o subgesture reasoning</th>
<th># Correct Detections w/ subgesture reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>19</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>Total</td>
<td>212</td>
<td>271</td>
</tr>
</tbody>
</table>

close to 1 most of the time. Figure 6-11 shows that this is the overall trend; however, closer examination reveals that the direction cosine is less than 0 (or the angle is larger than 90) for 3.6% of the training samples. The digit training set consists of a total of 17,629 transitions. This is not an insignificant number if the goal is to build a pruning classifier. On the other hand the angle cosine is probably reliable enough to be part of the matching cost as in [127]. The second classifier we considered is the position displacement classifier. Figure 6-12 shows a histogram of the displacement between two successive detections. The important information in this figure is that there is a maximum displacement of 32 pixels, meaning that in the training set the hand did not move more than 32 pixels. Since the correct threshold setting for this transition classifier is more evident compared to the single observation classifier, then rather than learning an expansion factor, we simply implemented a classifier that rejects a transition when the displacement is more than 40 pixels (which corresponds to adding an expansion factor of 8 pixels).
Figure 6.10: Fraction of visited cells (out of total DP cells) as a function of fraction of gesturing time (out of total time).

Figure 6.11: Histogram of cosine of the angle difference between the direction of the displacement vector between two successive detections and the estimated flow direction.

Figure 6.12: Histogram of displacements between consecutive hand locations.
6.4 ASL Retrieval Experiments

Sign Retrieval from ASL Utterances

In this experiment we consider a gesture-finding tool, where the system assists the user in efficiently finding occurrences of a gesture of interest in a long video sequence. In our case we want to find occurrences of an ASL sign in a video clip of an ASL story. The ASL story is narrated by an ASL native signer, which makes the temporal segmentation task more challenging mainly due to the fast signing and the wide variations in appearance within the same sign class.

In this experiment we find occurrences of 3 ASL signs: “BETTER”, “HERE”, and “WOW” in an ASL story. In that story there are 5 examples of the sign “BETTER”, 9 examples of “HERE”, and 4 examples of “WOW”. The story is 27,339 frames long at 30 frames per second. Given the model of the gesture of interest, we associate each frame of the input sequence with the matching cost of the optimal warping path going through that frame. Then, a threshold is chosen in a conservative way, by choosing the largest score associated with frames that belong to occurrences of the sign of interest. The user is asked to review all subsequences consisting of frames with scores below that threshold. The threshold is chosen in a way that guarantees that the user will see all occurrences of the sign of interest.

An example plot of the matching cost associated with each frame over time for the sign “HERE” is depicted in Figure 6-13. The two black strands shown in this figure are the frames corresponding to the two actual occurrences of the sign “HERE”. The four shaded strands represent all the input frames associated with matching costs that fall below the spotting threshold (represented by the black horizontal line). The user of the gesture finding tool will have to review only those frames instead of reviewing all the input frames.

Our performance measure is the fraction of frames that the user has to review over the total number of frames. That fraction reflects the time that the user has to spend. Naturally we want that fraction to be as low as possible. In the absence of a gesture-finding
Figure 6.13: Finding the ASL sign “BETTER” in frames 1–2103 of an ASL story. The blue line is the matching cost as a function of time. The black strands are the ground truth temporal segments where the sign was performed. Note that local minima in the cost function are reached within those segments. The four shaded strands represent all frames belonging to some optimal warping path for which the matching cost is below a threshold. The threshold is represented by a black horizontal line. The frames below the threshold will have to be inspected by the user in order to accurately localize the sign. The percentage of red frames (102) out of the total number of frames (2103) in this case is 4.85%.

tool, the user would simply have to review the entire video sequence, so the fraction of reviewed frames to total number of frames would be 1.0. Using our gesture-finding tool, the performance measure for the signs “BETTER,” “HERE,” and “WOW” is 0.19, 0.11, and 0.07 respectively, which amounts to time savings of a factor between 5 and 14 compared to reviewing the entire video.

ASL Sign Retrieval from Video

In these experiments we collected video clips of ASL signs. In every clip a native ASL signer gestures the 24 signs which are listed in Table 6.6. Eight of the 24 signs are one-handed signs, and 16 of the 24 signs are two-handed signs. Each sign belongs to one of five types [11]. (See Figure 6.14(b), and Table 6.14(a)). Each sign starts with a particular hand shape and ends with a particular hand shape. (See Table 6.6, and Figures 6.15 and
We collected 10 sequences where the signer wears two colored gloves: a green glove on the dominant (right) hand, and a purple glove on the non-dominant hand (left). Those sequences were used for learning the sign models. In addition, we collected 10 sequences, where the signer wears a short sleeved shirt. Those sequences were used for testing the retrieval system. Information about the start and end of every sign was not available to the system during testing.

The results for the sign retrieval experiment are shown for the one-handed signs in Table 6.7 and for the two-handed signs in Table 6.8. The performance measures we used are the number of false alarms generated when the spotting threshold is set to detect all 10 sign occurrences (i.e., 100% detection rate), and the retrieval ratio, which is defined as the ratio between the number of frames retrieved using that threshold divided by the total number of frames (which is 32,060). A small retrieval ratio indicates better performance.

For the one-handed signs the number of false alarms is relatively low and the proportion of frames retrieved by the system out of the total number of frames is also relatively small. Three signs: “PAST,” “WOW,” and “TELL”, that are not shown in the table, generated too many false alarms. The reason for that is that those signs have little apparent (2D) motion and/or their learned models are too simple (contain only a few states). This enabled many irrelevant detections to match the models well and generate false alarms.

For the two-handed signs the number of false alarms and retrieval ratio are even lower compared to the one-handed signs. This is to be expected since the input has to match the trajectory constraints of both hands, and therefore there is less chance for accidental motion to match the two-handed sign model well. Four signs: “BIG,” “FINISH,” “MANY,” and “TOGETHER”, that are not shown in the table, generated too many false alarms for the same reason mentioned above (i.e., those signs have little apparent (2D) motion and/or their learned models are too simple).

The large number of false alarms motivated the use of a verification step for rejecting the bulk of those false alarms, without rejecting correct detections. For the two-handed
sign “NOW” there were originally 65 false alarms. The sign “NOW” is a Type 1 sign, where the two hands have the same handshape (“Y”), and are moving synchronously from top to bottom. (See Figure 6·16). The appearance of the hand is distinct and remains the same throughout the sign. In such cases, the appearance-based classifier described in Chapter 5 can be very useful for verification. We applied the verification classifier to all the spotted examples of the sign “NOW”, and set the normalized voting threshold to accept all the positive examples. At this threshold the number of false alarms decreased from 65 to only 12. This result is very encouraging, indicating that verification can play a significant role in improving spotting performance. We believe that applying appearance-based verification classifiers to other signs with stable hand appearances may improve the performance even more.

Another problem we discovered from the analysis of the results is the modeling of signs with repetitive or periodic motion. In our ASL spotting experiments there were five signs with repetitive motion: “WOW,” “CAR,” “MAYBE,” “RAIN,” and “WHAT”. The models for those signs were trained in the same way the other models were trained, without taking advantage of the periodic structure. Different training examples of a particular sign had different number of repetitions, which resulted in inferior models. We believe that modeling those signs correctly using a single period as the basic unit will improve spotting accuracy. This remains a topic for future investigation.
<table>
<thead>
<tr>
<th>Sign Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 0</td>
<td>One handed, <strong>not</strong> contacting the body</td>
</tr>
<tr>
<td>Type X</td>
<td>One handed, contacting the body (but <strong>not</strong> other hand)</td>
</tr>
<tr>
<td>Type 1</td>
<td>Two handed, both moving, synchronous or alternating movements</td>
</tr>
<tr>
<td>Type 2</td>
<td>Two handed, one active, one passive, <strong>same</strong> handshape</td>
</tr>
<tr>
<td>Type 3</td>
<td>Two handed, one active, one passive, <strong>different</strong> handshape</td>
</tr>
</tbody>
</table>

(a)

(b) **Figure 6.14:** Classification of Signs based on Number of Hands [11].
<table>
<thead>
<tr>
<th>Sign</th>
<th># of hands</th>
<th>Type</th>
<th>Handshapes</th>
<th>POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>1</td>
<td>0</td>
<td>5 &gt; 0</td>
<td>Conj.</td>
</tr>
<tr>
<td>ARRIVE</td>
<td>2</td>
<td>2</td>
<td>B-L/B-L</td>
<td>Verb</td>
</tr>
<tr>
<td>BIG</td>
<td>2</td>
<td>1</td>
<td>bent-L/bent-L</td>
<td>Adj.</td>
</tr>
<tr>
<td>BORN</td>
<td>2</td>
<td>2</td>
<td>B-L/B-L</td>
<td>Verb</td>
</tr>
<tr>
<td>CAR</td>
<td>2</td>
<td>1</td>
<td>S/S</td>
<td>Noun</td>
</tr>
<tr>
<td>DECIDE</td>
<td>2</td>
<td>1</td>
<td>F/F</td>
<td>Verb</td>
</tr>
<tr>
<td>DIFFERENT</td>
<td>2</td>
<td>1</td>
<td>1/1</td>
<td>Adj.</td>
</tr>
<tr>
<td>FINISH</td>
<td>2</td>
<td>1</td>
<td>5/5</td>
<td>Verb</td>
</tr>
<tr>
<td>HERE</td>
<td>2</td>
<td>1</td>
<td>B-L/B-L</td>
<td>Adverb</td>
</tr>
<tr>
<td>KNOW</td>
<td>1</td>
<td>X</td>
<td>B-L</td>
<td>Verb</td>
</tr>
<tr>
<td>MAN</td>
<td>1</td>
<td>X</td>
<td>5</td>
<td>Noun</td>
</tr>
<tr>
<td>MANY</td>
<td>2</td>
<td>1</td>
<td>S &gt; 5</td>
<td>Adj.</td>
</tr>
<tr>
<td>MAYBE</td>
<td>2</td>
<td>1</td>
<td>B-L/B-L</td>
<td>Adverb</td>
</tr>
<tr>
<td>NOW</td>
<td>2</td>
<td>1</td>
<td>Y/Y</td>
<td>Adverb</td>
</tr>
</tbody>
</table>
| OUT      | 1          | 0    | 5 > 0          | Adj. (Verb)
| PAST     | 1          | 0    | B-L            | Adverb|
| RAIN     | 2          | 1    | bent 5/ bent 5 | Verb  |
| READ     | 2          | 3    | 2/B-L          | Verb  |
| TAKE-OFF | 2          | 2    | B-L/B-L        | Verb  |
| TELL     | 1          | 0    | 1              | Verb  |
| TOGETHER | 2          | 1    | A/A (10/10)    | Adverb|
| WHAT     | 2          | 1    | 5/5            | Q-word|
| WOW      | 1          | 0    | bent 5         | Expl. |
| YESTERDAY| 1          | X    | A(10)          | Adverb|

Table 6.6: List of signs. From left to right: sign name, number of hands used in signing, sign type based on Battison [11] (see Figure 6.14(b)), hand shapes (see Figures 6·15 and 6·16), and part of speech (POS). In the Handshapes column the start and end shapes are depicted as start > end, and dominant hand and non-dominant hand are depicted as dominant | non-dominant.

<table>
<thead>
<tr>
<th>Sign</th>
<th># False Alarms</th>
<th>Retrieval Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>26</td>
<td>1/51</td>
</tr>
<tr>
<td>KNOW</td>
<td>6</td>
<td>1/149</td>
</tr>
<tr>
<td>MAN</td>
<td>47</td>
<td>1/22</td>
</tr>
<tr>
<td>OUT</td>
<td>2</td>
<td>1/138</td>
</tr>
<tr>
<td>YESTERDAY</td>
<td>0</td>
<td>1/112</td>
</tr>
</tbody>
</table>

Table 6.7: Results of Spotting One-Handed Signs. The reported number of false alarms is for 100% detection rate. The retrieval ratio is the fraction between the number of frames retrieved by the system and the total number of frames.
Table 6.8: Results of Spotting Two-Handed Signs. The reported number of false alarms is for 100% detection rate. The retrieval ratio is the fraction between the number of retrieved frames and the total number of frames. For the sign “NOW” the number in boldface is for spotting with verification. All the other numbers are for spotting without verification.

<table>
<thead>
<tr>
<th>Sign</th>
<th># False Alarms</th>
<th>Retrieval Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRIVE</td>
<td>0</td>
<td>1/139</td>
</tr>
<tr>
<td>BORN</td>
<td>4</td>
<td>1/137</td>
</tr>
<tr>
<td>CAR</td>
<td>0</td>
<td>1/64</td>
</tr>
<tr>
<td>DECIDE</td>
<td>7</td>
<td>1/120</td>
</tr>
<tr>
<td>DIFFERENT</td>
<td>0</td>
<td>1/572</td>
</tr>
<tr>
<td>HERE</td>
<td>1</td>
<td>1/47</td>
</tr>
<tr>
<td>MAYBE</td>
<td>13</td>
<td>1/29</td>
</tr>
<tr>
<td>NOW</td>
<td>65 &gt; <strong>12</strong></td>
<td><strong>1/78 &gt; 1/185</strong></td>
</tr>
<tr>
<td>RAIN</td>
<td>35</td>
<td>1/48</td>
</tr>
<tr>
<td>READ</td>
<td>0</td>
<td>1/159</td>
</tr>
<tr>
<td>TAKE-OFF</td>
<td>0</td>
<td>1/324</td>
</tr>
<tr>
<td>WHAT</td>
<td>8</td>
<td>1/77</td>
</tr>
</tbody>
</table>
Figure 6.15: ASL Signs I: S means gesture Start and E means gesture End.
Figure 6-16: ASL Signs II: S means gesture Start and E means gesture End.
Chapter 7

Discussion and Future Work

This chapter starts with a summary of the three main contributions of this thesis: the spatiotemporal matching algorithm, the pruning algorithm, and the spotting algorithm, which includes reasoning about nested patterns. The following section describes the major strengths and current limitations of the proposed system. The major strength of the proposed system is its ability to function robustly in realistic environments with very few constraints imposed on the user or environment. The current limitations of the system are coarse hand localization which may lead to poor feature extraction and matching, and a search space that can become large when as the number of candidates increases. The last section describes how those problems may be addressed in future work.

7.1 Contributions of the Thesis

This section provides a summary of the main contributions made in this thesis: a spatiotemporal matching algorithm that extends dynamic programming formulation to accommodate multiple feature vectors at every time instance, a pruning method that uses model-specific classifiers to reject unlikely dynamic programming hypotheses, and a spotting algorithm that incorporates reasoning about nested patterns.

Bottom-up Top-Down

A key feature of our system is that it combines cheap low-level features with high-level model information to simultaneously localize the gesture in the image and recognize it
as belonging to one of a predefined gesture classes. The ability of the high-level model to handle multiple feature vectors at every frame, enables the use of low-level features that are cheaper to compute. This is important for current vision-based systems that spend most of the time on low-level image processing for detection and/or segmentation. This comes at a cost of a more sophisticated and more expensive higher level matching component. To alleviate the burden of the matching process a principled pruning method is employed with a good tradeoff between speed and accuracy.

**Spatiotemporal Segmentation**

The most important contribution of this thesis at the application level is the ability to segment the gesture both in space and time. Previous algorithms assumed either very accurate spatial segmentation or very accurate temporal segmentation, but the algorithm proposed in this thesis assumes neither. Given a long video stream the proposed system can simultaneously locate the hand in every frame of the image sequence (spatial segmentation), and find the gesture start and end frame (temporal segmentation).

**Learning to Prune**

The goal of pruning in general is to cut down on the solution search space in order to reach the desired solution faster. It was noticed by other researchers [58] and verified in our experiments that pruning in dynamic programming-based, in addition to reducing the search time, may also improve accuracy. Improving both speed and accuracy at the same time is somewhat counter intuitive. The explanation is that pruning may remove degenerate alignments that happen to have better matching costs. We note however that this is not always the case.

One of the most popular pruning techniques for Viterbi search is the beam-search. The beam-search maintains promising hypotheses that have low matching costs within a beam width from the current best hypothesis. The beam width is typically set in an ad-hoc way. In this thesis, we proposed to view pruning as a binary classification problem, and
use machine learning methods to learn the beam width from training data. In addition, traditional beam-search uses the cumulative matching cost to prune unlikely hypotheses. In contrast, we propose that the features and similarity measures that are represented in the classifiers and are used for pruning can be different than those that are used for matching and recognition. Furthermore, any type of binary classifier can be used for this purpose. In our experiments we used simple decision stumps.

In addition to the cumulative cost based classifiers we proposed two additional types of classifiers: single observation classifiers and transition classifiers. The single observation classifiers can prune hypotheses when a single observation matches a particular model state poorly. This may raise the concern of pruning an entire trajectory based on a single outlier observation. We note that a single outlier observation in traditional DTW or HMM models can significantly reduce the likelihood of an entire trajectory, therefore having a similar net effect, so it might be still beneficial to prune early on. Second, we note that as we increase the number of candidate feature vectors at every time step the chance of having an outlier observation decreases.

Generally speaking, the performance of any classifier improves as the number of training examples increases, assuming that the distribution of training examples is representative of the true distribution. In our experiments there were 30 training examples per class in the digits experiment, and 10 training examples per class in the ASL experiment. This is a relatively small number of training examples, and we expect that the quality of pruning classifiers increase with the training set size.

The pruning method that has been proposed in this thesis is applicable when there is a single feature vector or multiple feature vectors. The benefit from pruning in the case of multiple feature vectors is expected to be even better than in the case of a single feature vector since the proportion of hypotheses that are likely to match well is expected to be smaller.
Subgesture Reasoning and Learning

If a particular gesture matches well the prefix of another gesture, then if the input sequence consists of one of those gestures, then at some point in time when the endpoint of the common prefix has been reached, the matching costs of both models will be similar and relatively small. A lazy evaluation recognition system will classify the input to the shorter gesture class because its model has been completely matched. However, future input frames may reveal that the longer gesture has actually been performed. To increase the chance of making the correct decision we postpone the decision, and monitor the matching of the longer gesture model. If at a later point the model of the longer gesture is matched completely with a subsequence of the input then the longer gesture is recognized. Otherwise, the shorter gesture will be recognized.

The proposed method for learning subgestures matches a pair of gestures using DTW: the hypothesized subgesture model and the hypothesized supergesture query. One approach for detecting a subgesture would be to conceptually plot the cumulative matching cost of the last model state over time and look for a significant minimum in that cost, but this would require a threshold setting. Instead we proposed to use the pruning information in the matching, and then decide that a subgesture relation exists if there exist a valid warping path passes through unpruned DP cells and connects the first and last model states. This removes the requirement of a threshold setting, and as a side effect it is easier to detect the existence of a warping path in the pruned DP table.

In our current system, we learned subgesture relations between complete gestures. However, subgestures or rather gesture segments can be used to define sequential subunits, similar to phonetic acoustic models in speech [83], making every gesture a concatenation of segment models. This can be useful for reducing the amount of training data and for scaling the recognition system to large vocabularies, because the number of gesture segment models is significantly smaller than the number of gestures in large vocabulary recognition systems. In addition, new gesture models need not be fully trained if they can be constructed using the learned segment models.
A number of researchers have defined subunits for the purpose of sign language recognition. Bauer and Kraiss [12] used self-organizing maps to learn the parameters of subunit models, while Vogler [115], Yuan et al. [126] and Wang et al. [117] used linguistic knowledge to define the subunits instead of using unsupervised learning. We believe that our method for learning subgesture relations between complete gestures can be extended to automatically identify a compact set of gesture segment models and to learn the parameters of those models.

**Verification**

The dynamic programming framework used in the search for the optimal alignment between the input and the model can sometimes lead to the wrong solutions. Dynamic programming is a divide and conquer approach, where global problem is broken up into a sequence of local subproblems. The solution of each subproblems is straightforward. In our context, it is simply finding the hypothesis with the minimum matching cost among a few neighboring hypotheses. The solutions to those subproblems are then combined to form the solution to the global problem. In our context, it is simply adding the local matching costs. The global optimal solution is therefore dependent on the local features and similarity measure used for computing the local matching cost.

Many factors including noise, occlusion, and illumination changes may lead to changes in the features extracted from the input sequence, and consequently lead to a global optimal solution which is different than the desired solution. The hypothesized solution can be passed through a verifier that decides whether to accept or reject the claimed identity of the hypothesis. Verification can be carried out using additional more expensive features and similarity measures that would otherwise be too expensive to compute for all the hypotheses in the matching stage. In the experiments we have shown how hand appearance can be used to effectively reject false matches of gestures that are similar to a particular model in terms of trajectory but are quite different in hand shape and appearance. In general, we believe that a good strategy for verification is to use classifiers which are based on features
which are different than the features which are used for matching.

7.2 Strengths and Limitations

The experimental results have shown that the gesture spotting and recognition system performs well under challenging imaging conditions. However, the performance of the algorithm depends on several factors including the number of candidates used in the algorithm, the quality of hand localization, the features that are used for matching, and the actual models used to represent the gestures. This section starts with enumerating the imaging conditions under which the current system is expected to perform well, followed by a discussion on the effects that the abovementioned factors may have on the system’s performance.

Removing Imaging Restrictions Used in Gesture Recognition Systems

In their recent survey Ong and Ranganath [82] enumerate (in Table 1) the imaging restrictions and constraints used in vision-based approaches. We repeat all of these assumptions here:

1. long-sleeved clothing
2. colored gloves
3. uniform background
4. complex but stationary background
5. head/face required to be stationary or have less movement than hands
6. constant movement of hands required
7. fixed body location and pose specific initial hand location
8. left hand and/or face excluded from field-of-view
9. vocabulary restricted or unnatural signing to avoid overlapping hands or hands over face

10. field-of-view restricted to the hand which is kept at fixed orientation distance to camera

It is quite remarkable that with the simple incorporation of data association into the matching process the vast majority of those restrictions can be removed. In our experiments the user wore short sleeves. Colored gloves were only used for training the gesture models, but were not used in the test sequences. The background in the digits experiments was a cluttered office background. In the ASL experiment the background was uniform but this is not required.

In the experiments, the face had less movement than the hands most of the time, but occasionally the face was faster than the hands, and though the best candidate (with respect to detection score) corresponded to the face, the matching algorithm could locate the candidate corresponding to the hand as the best in terms of matching score. Constant movement of the hands was not required. The reason is that though the detection of the hands is based on motion in addition to color, in contrast to many other approaches, no thresholding was done on the difference image. Therefore, even if the hand is stationary image noise can cause spurious motion which together with the skin likelihood image will result in detection.

The requirement of pose specific initial hand location is certainly not required. Both hands and face may or may not be included in the field of view, without affecting the correct behavior of the system. Overlapping hands are not a problem. If a single candidate corresponds to both hands it is not a problem for a single handed gesture, neither it is a problem for a two-handed gesture because the candidate feature vector can be duplicated for both hands. The field of view is not restricted to the hand. The hand occupies only a small portion of the field of view. This is currently a big problem for extracting meaningful appearance-based features for recognition. Even for higher resolution hand
images, appearance-based matching of hand images is still an open problem.

Finally, we do implicitly assume that the hand is at a fixed distance from the camera. A more accurate statement is that the size of the hand subimage is an input parameter to the system. Assuming a frontal face detector the expected hand image size can be set relative to the face image size. Furthermore, if appearance-based features are not used, but only trajectory features are used, then the image size is less important. However, there is a tradeoff: a smaller image patch will cover part of the hand but require more candidates, and a larger image patch will likely include non-hand pixels but require fewer candidates.

**Geometric Invariance**

Our current system is translation dependent, that is, it assumes that gestures are performed in a particular location in the image. Using translation invariant features such as velocity or direction may lead to degraded recognition performance, as shown in the experiments. To address this problem, we showed, in Chapter 3 Section 3.5, how to incorporate translation invariance in the isolated gesture recognition case. The basic idea is to run multiple matching processes, each one corresponding to a different hand location in the first frame, and to subtract that location from all subsequent detections. In the continuous gesture recognition case, the gesture can start at any time instant, and therefore running a separate matching process for every possible hand location and start frame hypothesis may be too computationally demanding.

Another alternative for obtaining translation invariance is to measure hand locations relative to a fixed body part, which can be reliably detected, such as the shoulders or the face. The size of the face detection window or the distance between the shoulders can also be used to fix the scale. Another source of evidence which can be used to fix the scale is the gesturing motion envelop, which can be determined from simple motion cues. However, gestures may be performed at different locations and in different scales even with respect to another body part. This is true for the digit recognition application. For the sign spotting application the situation is better because in sign language a sign is defined by its location
Another way that is often used to normalize the input with respect to scale is to “box” the current input segment by resizing its spatial extent to a fixed window size [9, 17]. This assumes that a partial trajectory has already been extracted. In our system, a partial trajectory can be reconstructed from backtracking the DP table. To reduce computational load backtracking can be performed only once in every few frames. Yet another approach is to learn different models with different scales, however this approach is not very scalable. Instead of representing models at different scales, the input can be represented at different scales using image pyramid techniques [1], and then multiple input images at different scales can be matched with a single model in a normalized scale. Overall, incorporating translation and scale invariance in the gesture spotting system would free the user from additional gesturing constraints, and would make the system more generally applicable. We are therefore motivated to address this problem in future work.

**Hand Localization**

Unreliable and inaccurate hand localization was the main motivation for developing the spatiotemporal matching algorithm. It lead to a spatiotemporal matching algorithm that can accommodate multiple hand hypotheses, and that implicitly assumes that one of the hand location hypotheses leads to accurate recognition. Hand localization may be sufficiently accurate for detection purposes, but not accurate enough for matching and recognition.
purpose.

Despite recent advances in hand localization [81, 63] those appearance-based methods typically do not scale very well. The reason is that every different hand appearance gives rise to a different detector that has to be scanned over the input image. Using appearance-based methods it is currently hard to imagine a general hand detector that would be sufficiently accurate over a wide variation in hand appearance.

Even if we had access to the “best” rectangular region that corresponds to the hand, correct matching would still not be trivial. There are several reasons for that. First and foremost, the hand does not project to a rectangular region, nor can it be sufficiently well approximated by a rectangular region. If the region is too small then it would only cover part of the hand. On the other hand, if the region is sufficiently large it will include in addition to hand pixels also non-hand background pixels. Those background pixels have a significant effect on image similarity measures such as correlation, chamfer matching, shape context matching, and edge orientation histogram matching. Finally, small changes in hand pose or shape can lead to significant change in matching cost. Furthermore, hand images of two different users with the same hand pose and shape may appear significantly different due to anthropomorphic differences.

**Dependence on Number of Candidates**

As the number of candidates increases the chance of including the target object in one of those candidates increases. Therefore, in theory, the classification error should not increase. However, in practice we observed in the first isolated digit experiment that the classification error decreases up to a particular number of candidates, but then it starts increasing due to false matches. The reason for that is that either the extracted features and/or the similarity measure used for matching are not powerful enough for discrimination. The optimal number of candidates that should be used therefore depends on the similarity measure, but it also depends on the complexity of scene, namely how many objects are moving and how cluttered is the background. For the similarity measures and types of
scenes used in our experiments the optimal number of candidates $K^*$ was between 2 and 5.

**Complexity**

Compared to DTW and CDP our basic spatiotemporal matching algorithm without pruning requires a factor of $K$ more memory, and is $K$ times slower. If transition costs or transition-based classifiers are incorporated then an additional factor of $K$ penalty is incurred on the time requirement. If the gesture or motion pattern to be recognized consist of $n$ interacting objects, then the space and time requirements increase exponentially by a factor of $K^n$. This exponential increase is typical in data association problems. Fortunately, in practice, the number of interacting objects is small: two hands in our gesture recognition scenario. Furthermore, our pruning method cuts down on the number of matching hypotheses that need to be propagated from frame to frame, and therefore speeds up the matching algorithm.

**Features for Recognition**

In the experiments we observed that 2D position features are pretty reliable for the digit recognition experiments. In the ASL experiments we observed that the location of a sign is not as well defined as might be inferred from Sign Language notation systems. Although the sign cannot appear in arbitrary position with respect to the body, the variation in the location of signs can be rather significant. The success of a sign language recognition system will depend on the the amount of training data available and the variation in the training data.

**Dynamic Model**

We observed that for the digit dataset a model with a small number of states, where the observation density of each state is a simple Gaussian, is not a good approximation of the digit trajectories. The reason is that two points on the input trajectory, that should be
equally likely given a particular state, could be penalized differently depending on their normalized distance from the Gaussian. A better way to represent a trajectory is using a segmental model. Instead of representing a state observation density with a simple Gaussian, in a segmental model the observation density of each state is an entire segment (e.g., a line, cubic spline) of observations.

### 7.3 Potential Applications

The spatiotemporal matching method presented in this thesis can be applied to problems which require solving a temporal matching problem and a data association problem simultaneously. One potential application is surveillance, where people can move along a specified number of paths which can be represented by trajectory models. Person or pedestrian detection algorithms can generate candidate hypotheses for every frame, and spatiotemporal matching algorithm can predict when a person moved along a particular trajectory. Different models can be learned for the different trajectories along which people are allowed to walk, and the system would alert the operator in cases where a person deviates from any of those trajectories.

Another potential application is in speech recognition, in cases where there are multiple speakers, and the goal is to recognize the utterances of a particular individual. If the speech signal can be decomposed into multiple candidate features every frame then the spatiotemporal matching framework can be employed.

Another area where the 3D Dynamic Programming matching algorithm has been applied recently is the detection of shapes with variable structure in images [8]. In the proposed method there is a single shape model, where every state can either generate a part of the shape or not. The multiple feature candidates are simply image edge pixels. The DP algorithm tries to find the set of edge features that best explain a particular shape structure.
7.4 Future Work

The main motivation for the methods proposed in this thesis was to make gesture spotting and recognition systems useful in a wider range of applications, which impose very few constraints on the users or the environment. We believe the proposed system is an important step towards meeting that goal, however there is still much room for improvement. In order to improve and extend our current system we need to address some of the weaknesses brought up in the previous sections. In order to improve accuracy a better hand localization method is needed. Better hand localization will lead to extraction of more meaningful hand features and more meaningful matching scores. One way to improve hand localization is by incorporating a hand tracker in the framework. In order to speed up the search for the best matching gesture even more, the gestures can be broken up into smaller subunits, and the search can be done over the smaller space of subunits. The challenge here is how many subunits to use, and which models to use for those subunits.

Incorporating a Tracker

The gesture recognition algorithm proposed in this thesis does not require explicit tracking. Consequently, the two main problems of visual tracking - tracking initialization and recovery from tracking failure - are avoided. Both of these problems are related to the typical high dimensionality of the tracker's state and to the non-linearity of the tracker’s measurement model, which can be non-linear measurement model. This means that a small change in the current estimate can lead to a significantly different next estimate. In our approach these problems are avoided by maintaining multiple hypotheses in every frame. Therefore, no careful initialization is required in the first frame, assuming that one of the hypotheses corresponds to the tracked object. For the same reason, tracking failure is not an issue because the underlying gesture model will match one of its states with one of the candidate hypotheses.

The idea of motion consistency that is typically employed in tracking models can be
modeled in our framework through transition classifiers. In practice we found that the
classifier based on absolute displacement can be useful in pruning unlikely hypotheses.
The reason is that in a given scene the hand cannot move more than a certain number of
pixels. On the other hand, we found that a reliable pruning decision cannot be done by
a transition classifier based on the consistency between the estimated flow direction and
the direction between successive detection windows. The reason is that neither the flow
direction nor the direction between successive detection windows are sufficiently reliable
for making a hard pruning decision. At the same time, the consistency score is sufficiently
reliable to be part of the overall matching score as was done in [127], and leads to improved
classification accuracy.

Notwithstanding the advantage of not needing to track the gesture, incorporating a
tracker in this framework will probably improve the localization accuracy, which conse-
quently improve the feature matching accuracy, which will finally improve the recognition
accuracy. The problems with incorporating such a tracker are similar to the problems that
emerge in multiple hypothesis tracking. That is, how to decide which hypotheses are worth
tracking and which hypotheses can be pruned from further consideration. Furthermore,
how many tracks to maintain, and how to decide when to split or merge different tracks.

The Gesture Search Space

Successful human computer interfaces require the user to neither memorize nor navigate
through a large number of commands. The size of the basic command set is typically small.
For example, a typical web browser has a handful of navigation buttons, and a typical
window-based application has a few tens of menu items to choose from. Similarly, a pen-
based interface enables the user to interact with the device through use of alphanumeric
pen gestures, with an alphabet size of a few tens. Some interfaces even allow the user to
define the set of gestures he or she wants to use for interaction [17]. For vision-based gesture
interfaces the situation is similar. The gesture vocabulary is typically small: in current
systems vocabulary sizes range between a handful and a few tens of gestures. For those
small to moderate size vocabularies the methods presented in this thesis are applicable. The method search time of the method is linear to the number of gesture models, and when this number is small there is no real justification for a sublinear search method.

The situation is different in text retrieval, sequence alignment in bioinformatics, and speech recognition where vocabulary sizes are relatively large. For example, a large vocabulary continuous speech recognition (LVCSR) system may consist of a few tens of thousands of words. To make the search time in LVCSR practical subword units are used instead of word units. The number of subword units is substantially smaller than the number of words, typically a few tens, and all the words can be composed of the subword units. All the words can be arranged in a tree structure, where every branch represents a subword unit and a particular word can be recovered by following the path from the root to a leaf. The tree structure allows to prune early on a large number of unlikely word hypotheses that consist of subword units that match poorly with the input acoustics.

In the digit recognition application digits can be represented by a sequence of subunits which are curve segments. Different types of segment models can be used including linear segments, quadratic segments or more generally any p-degree polynomial segments, or splines. To make sign language more scalable researchers have used movement and hold segments as subunits. Those methods were applied to glove data. In our ASL video dataset the signs are performed in natural speed by ASL signers, and only a few tens of frames are available for every sign at 60 frame per second. It is hard even for a human to segment the sign into meaningful subunits.

Related to the problem of modeling signs with subunits is modeling signs with repetitive or periodic motion. In our ASL spotting experiments there were five signs with repetitive motion: “WOW,” “CAR,” “MAYBE,” “RAIN,” and “WHAT”. The models for those signs were trained in the same way the other models were trained, without taking advantage of the periodic structure. Some training examples of a particular sign appeared with a different number of repetitions than other examples, which resulted in inferior models. We believe that modeling those signs correctly using a single period as the basic unit will
improve the spotting performance even more.
Chapter 8

Conclusion

In this thesis, we presented a framework for spotting and recognizing gestures in continuous video streams. The proposed algorithm simultaneously localizes the gesture in every frame of the video (spatial segmentation), as well as detect the start and end frames of the input gestures (temporal segmentation).

Most existing methods [75, 67, 56, 129] assume that a single feature vector that corresponds to the moving object can be extracted from every input frame reliably. We argue that this assumption does not hold for many practical applications. For example, in hand tracking and recognition applications hand detection may be unreliable for a number of different reasons, including cluttered background, multiple moving objects in the scene, hand face occlusions, and certain clothing (patterned and/or short-sleeved shirt). The proposed framework is designed to function reliably under such circumstances by accommodating multiple feature vectors at every time instance during the matching process.

We believe that future work should focus on improving hand localization which will consequently lead to extraction of more useful features for matching. In addition, there is still more room to cut down on the search space by adaptively choosing fewer candidates in every frame, and by modeling the gestures with smaller segment units.

The major contributions of this thesis are:

1. A dynamic programming based algorithm for matching between model pattern(s) and an input stream that may consist of multiple input observations at every time instance.
2. An efficient dynamic programming search method that uses pruning classifiers to prune unlikely DP search hypotheses, and an algorithm for learning the parameters of those pruning classifiers.

3. A method for spotting patterns in an input stream, that can correctly handle patterns which are nested within other patterns, and that uses verification to reject patterns which were falsely matched.

Experiments with our proposed gesture spotting and recognition system have demonstrated that accommodating multiple input feature vectors in the matching almost always improves accuracy compared to using only the best feature vector at every time instance, with accuracy gains of up to 15%. The proposed pruning method speeds up the matching procedure by up to a factor of ten, and in some cases even improves the accuracy. Finally, our method for reasoning about nested patterns can avoid many of the false detections that occur when an input pattern contains another pattern as a subpart, and thus improve the system’s accuracy by up to 10%.

The proposed methods can be applied in other areas, including speech and bioinformatics, where dynamic programming based matching algorithms and pattern registration methods are employed. In the context of vision-based gesture recognition systems, we believe that the methods proposed in this thesis advance the state-of-the-art in that systems that employ the proposed methods can be applied to a wider range of applications and function reliably in more realistic environments removing many of the constraints that are often imposed on the user or environment.
References


CURRICULUM VITAE

Jonathan Alon

Computer Science Department  
Boston University  
Boston, MA 02215  
Email: jalon@cs.bu.edu  
Phone: 617-353-1498

Research Interests

Computer Vision, Pattern Recognition, Motion-based Recognition, Gesture Recognition, Human Computer Interfaces, Similarity Indexing, Image and Video Databases.

Education

September 2001 - Present  Ph.D. Student, Computer Science Department, Boston University. Advisor: Prof. Stan Sclaroff.

September 1997 - August 2001  M.A. in Computer Science, Computer Science Department, Boston University. Advisor: Prof. Stan Sclaroff.

September 1991 - June 1994  B.Sc. in Physics and Computer Science, Faculty of Exact Sciences, Tel-Aviv University. Israel.

Publications

Refereed Conferences


Refereed Workshops


Research Experience

- Computer Science Department, Boston University, Boston, MA. 09/1997 - Present. Research Assistant for Prof. Stan Sclaroff.
  
  - Designed Dynamic Space-Time Warping (DSTW), a gesture recognition method that combines low-level features and high-level model information to simultaneously recognize the gesture and localize the hand in the video sequence.

  - Designed a method for fast approximate alignment between objects that are represented as sequences of features (like time series) or unordered sets of features (like edge images) based on feature correspondences between each object and a small set of prototype objects.

  - Designed (with Vassilis Athitsos) BoostMap, a machine learning method for constructing Euclidean embeddings that are optimized for similarity indexing or nearest-neighbor classification accuracy. BoostMap can be used with arbitrary non-Euclidean and non-metric distance measures, in domains including edge images, temporal sequences, or biological databases.
  Research Intern, supervised by Paul Beardsley.
    – Created a module that performs 3D scanning of objects using multiple non-
      overlapping cameras. The main focus of the work was on the recovery of camera
      parameters - relative position and orientation - as well as internal parameters,
      including lens distortion.

Teaching Experience

  Teaching Assistant for following courses:
    – Introduction to Computer Graphics
    – Introduction to Computer Science with Intensive C++
    – Introduction to Computers

• “Ironi Tet” high school. Math Instructor of ninth grade.

Professional Activities

• Reviewer for the following conferences: ICCV 2005, CVPR 2005, CIVR 2005, CVPR

• Reviewer for the following workshops: ACMMM VSSN 2005, ICCV HCI 2005, PETS