Multiscale Dynamic Time and Space Warping

by

Fitriani

B.Eng., Computer Engineering
Nanyang Technological University, 2007

Submitted to the School of Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Author .....
School of Engineering
August 14, 2008

Certified by .....
Brian W. Anthony
Research Scientist, Laboratory for Manufacturing and Productivity
Director SMA–Manufacturing Systems and Technology
Thesis Supervisor

Accepted by .....
Jaime Peraire
Professor of Aeronautics and Astronautics
Codirector, Computation for Design and Optimization Program

ARCHIVES

Multiscale Dynamic Time and Space Warping

by

Fitriani

Submitted to the School of Engineering
on August 14, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Computation for Design and Optimization

Abstract

Dynamic Time and Space Warping (DTSW) is a technique used in video matching applications to find the optimal alignment between two videos. Because DTSW requires $O(N^4)$ time and space complexity, it is only suitable for short and coarse resolution videos. In this thesis, we introduce Multiscale DTSW: a modification of DTSW that has linear time and space complexity ($O(N)$) with good accuracy.

The first step in Multiscale DTSW is to apply the DTSW algorithm to coarse resolution input videos. In the next step, Multiscale DTSW projects the solution from coarse resolution to finer resolution. A solution for finer resolution can be found effectively by refining the projected solution. Multiscale DTSW then repeatedly projects a solution from the current resolution to finer resolution and refines it until the desired resolution is reached.

I have explored the linear time and space complexity ($O(N)$) of Multiscale DTSW both theoretically and empirically. I also have shown that Multiscale DTSW achieves almost the same accuracy as DTSW.

Because of its efficiency in computational cost, Multiscale DTSW is suitable for video detection and video classification applications. We have developed a Multiscale-DTSW-based video classification framework that achieves the same accuracy as a DTSW-based video classification framework with greater than 50 percent reduction in the execution time. We have also developed a video detection application that is based on Dynamic Space Warping (DSW) and Multiscale DTSW methods and is able to detect a query video inside a target video in a short time.

Thesis Supervisor: Brian W. Anthony
Title: Research Scientist, Laboratory for Manufacturing and Productivity

Acknowledgments

I would like to thank my thesis advisor, Dr. Brian W. Anthony, for his invaluable advice and guidance during my research. Throughout my research project, he has painstakingly imparted his knowledge in the area of this research. His ideas and suggestions on the design of Multiscale DTSW and its applications were very valuable. Without his full support and great patience, it would be impossible for this research to be such a fulfilling and enriching experience.

I would also like to thank Singapore-MIT Alliance (SMA) for providing the required funding for my Master degree. I would like to specially thank Dr. John Desforge and Jocelyn Sales for their assistance and attention to every detail of our stay. I would also like to extend my gratitude to Michael Lim for helping me with the administration stuff.

I would also like to thank Computation for Design and Optimization Program for providing the great opportunity to carry out this research. I would like to specially thank Laura Koller for her assistance on all the administrative procedures.

Finally, I would like to thank my parents and all my friends who have given me strength and encouragement throughout the research.

Contents

1 Introduction **23**

1.1 Motivation 23

1.1.1 Video Query Application 23

1.1.2 Video Detection Application 23

1.1.3 Video Classification Application 24

1.1.4 Combination of Video Classification and Detection Application 25

1.1.5 Video Comparison Based Judging 25

1.2 Background 26

1.3 Problem Statement 26

1.4 Contributions 27

1.5 Literature Review 27

1.6 Document Outline 29

2 Related Algorithms **31**

2.1 DTW 31

2.1.1 DTW Algorithm 31

2.1.2 Complexity of DTW 33

2.1.3 FastDTW 34

2.2 DTSW 35

2.2.1 DTSW Algorithm 35

2.2.2 Complexity of DTSW 38

3 Multiscale DTSW **41**

3.1 Multiscale DSTW Algorithm 44

3.2 Complexity of Multiscale DTSW 51

3.2.1 Time Complexity 51

3.2.2 Space Complexity 57

3.3 Analysis of Multiscale DTSW Algorithm 58

3.3.1 Optimal Level 60

3.3.2 Relaxation Radius 61

3.3.3 Multiscale DTSW Efficiency 65

3.4 Experimental Result 67

4 Extension of Multiscale DTSW **77**

4.1 Multiscale DTSW with Eigenframes Implementation 77

4.1.1 Principal Component Analysis 77

4.1.2 Implementation of Eigenframes in Multiscale DTSW 78

4.1.3 Experimental Result 81

4.2 Multiscale DTSW with Control Points 87

4.3 Multiscale DTSW with Level Jump 89

4.4 Piece-wise Multiscale DTSW 93

5 Multiscale DTSW in Video Classification Application **97**

5.1 Video Classification Application 97

5.2 Selection of Decision Variables 98

5.2.1 Motion Scalar Combination 101

5.2.2 Resolution 105

5.2.3 Maximum Allowable Changes in the x and y Dimensions . . . 109

5.2.4 Variance for Multiscale DTSWEF 110

5.3 The Performance of the Video Classification Application 111

5.3.1 Comparison between DTSW, Multiscale DTSW, and Multiscale DTSWEF 111

5.3.2 Sensitivity to Scale Variance 113

6 Multiscale DTSW in Video Detection Application **115**

6.1 Video Detection Application 115

6.2 Methods for Comparing Videos in the Video Detection Application . 116

6.2.1 DSW 117

6.2.2 Multiscale DTSW 117

6.2.3 Combination of DSW and Multiscale DTSW 119

6.3 The Performance of the Video Detection Application 119

6.3.1 DSW-based Video Detection Application 119

6.3.2 Multiscale-DTSW-based Video Detection Application 121

6.3.3 DSW-and-Multiscale-DTSW-based Video Detection Application 122

6.3.4 Sensitivity to Scale Variance 125

7 Summary **127**

7.1 Contributions 127

7.2 Future work 128

A Target and query videos **129**

A.1 Karate punch videos 130

A.2 Heart valve videos 132

A.3 Horse racing videos 134

A.4 Person walking videos 136

A.5 Palm opening and closing videos 138

A.6 Random videos 140

A.7 Video classification template videos 142

B Piece-wise Multiscale DTSW on Karate Punch Videos **145**

List of Figures

3-1 Diagram of DTSW 42

3-2 Diagram of Multiscale DTSW with optimal level = 2, $r_s = 0$, $r_t = 0$. 43

3-3 Flowchart of Multiscale DTSW 47

3-4 Similarity at each level of Multiscale DTSW's execution in comparing the karate punch videos 48

3-5 Similarity at each level of Multiscale DTSW's execution in comparing the horse racing videos 49

3-6 Similarity at each level of Multiscale DTSW's execution in comparing the heart valve videos 49

3-7 Similarity at each level of Multiscale DTSW's execution in comparing the palm opening and closing videos 50

3-8 Projection of a horizontal warp path to finer resolution hypervolume with radius = 1 52

3-9 Projection of a vertical warp path to finer resolution hypervolume with radius = 1 52

3-10 Projection of a diagonal warp path to finer resolution hypervolume with radius = 1 53

3-11 Projection of a warp path to finer resolution hypervolume in the time dimension with $r_t = 1$ 53

3-12 Projection of a warp path to finer resolution hypervolume in the x dimension with $r_s = 1$ 55

3-13 Projection of a warp path to finer resolution hypervolume in the y dimension with $r_s = 1$ 55

3-14 Total computations versus number of levels of Multiscale DTSW in comparing two videos with the length of each dimension of the *Elemental Distance* and *Cumulative Distance* hypervolumes (N) equal to 100 and $r_s = r_t = \{5, 10, 15, 20, 25, 30\}$ 60

3-15 Optimal level versus relaxation radius of Multiscale DTSW in comparing two videos with N=100 66

3-16 Normalized time complexity of Multiscale DTSW versus relaxation radius in comparing two videos with N=$\{50, 100, 500, 1000\}$ 67

3-17 Normalized time complexity of Multiscale DTSW versus log N with relaxation radius = $\{0.5, 0.6, 0.7, 0.8\}$ 68

3-18 Execution time of Multiscale DTSW running on a PC with 3.4 GHz processor and 1 GB memory versus number of levels in comparing the karate punch videos with dimension = $81 \times 89 \times 26 \times 13$ and $r_t = r_s = \{5, 10, 15, 20\}$ 71

3-19 Execution time of Multiscale DTSW running on a PC with 3.4 GHz processor and 1 GB memory versus number of levels in comparing the karate punch videos with dimension = $81 \times 89 \times 26 \times 13$ and $r_t = r_s = \{25, 30, 35\}$ 72

3-20 Execution time of DTSW and Multiscale DTSW running on a PC with 3.4 GHz processor and 1GB memory in comparing the karate punch videos with $r_s = r_t = 5$ 74

4-1 Computing the *Elemental Distance* hypervolume in the DTSW algorithm 79

4-2 Computing the *Elemental Distance* hypervolume in the DTSW algorithm with Eigenframes implementation 80

4-3 The normalized error and normalized execution time in comparing the karate punch videos using Multiscale DTSW with Eigenframes implementation and various settings for the variance 83

4-4 The normalized error and normalized execution time in comparing the heart valve videos using Multiscale DTSW with Eigenframes implementation and various settings for the variance 84

4-5 The normalized error and normalized execution time in comparing the palm opening and closing videos using Multiscale DTSW with Eigenframes implementation and various settings for the variance 85

4-6 Known-points-based prediction of a warp path in the time dimension 87

4-7 Known-points-based prediction of a warp path in the space dimension 88

4-8 The optimal warp paths of comparing the karate punch videos found by using Multiscale DTSW at level = 1 to level = 6 90

4-9 Projection of all optimal warp pathes of comparing the karate punch videos found by using Multiscale DTSW at level=1 to level=6 onto a common axis 91

4-10 The top left figure shows an example of the optimal warp path at level=3. The top right figure shows an example of the optimal warp path at level=4. The bottom figure shows the projection of both optimal warp paths to a common axis. The difference between the two optimal warp paths is defined as the summation of absolute vertical differences between both warp paths projected to a common axis as shown in the shaded region in the figure. 92

4-11 Flowchart of Multiscale DTSW with Level Jump 94

5-1 The diagram of the video classification application 98

5-2 An example of a template video and the corresponding template source video 99

5-3 An example of Receiver Operator Characteristic (ROC) plot 100

5-4 ROC plot of the video classification application on the training set with different motion scalar combinations 102

5-5 ROC plot of the video classification application on the unknown set with different motion scalar combinations 102

5-6 ROC plot of the video classification application on the training set with different settings of the normalized motion scalar combination 104
5-7 ROC plot of the video classification application on the unknown set with different settings of the normalized motion scalar combination . 104
5-8 ROC plot of the video classification application on the training set using $u_N{}^2$ as the motion scalar combination and different sets of resolution 106
5-9 ROC plot of the video classification application on the unknown set using $u_N{}^2$ as the motion scalar combination and different sets of resolution 107
5-10 ROC plot of the video classification application on the training set using $\mathbf{u}$ as the motion scalar combination and different sets of resolution 108
5-11 ROC plot of the video classification application on the unknown set using $\mathbf{u}$ as the motion scalar combination and different sets of resolution 108

6-1 The diagram of the video detection application 116
6-2 Three categories of the optimal warp path in the time dimension of matching a target subvideo to the query video. (a) The target subvideo is too long to match with the query video (b) The target subvideo matches the query video (c) The target subvideo is too short to match with the query video. 118

A-1 Karate punch query video 130
A-2 Karate punch target video 131
A-3 Heart valve query video 132
A-4 Heart valve target video 133
A-5 Horse racing query video 134
A-6 Horse racing target video 135
A-7 A query video showing a person who is walking 136
A-8 A target video showing a person walking on the beach 137
A-9 Palm opening and closing query video 138
A-10 Palm opening and closing target video 139
A-11 Random query video 140

A-12 Random target video 141
A-13 A template video for each class in the video classification application database: walk, run, side, skip, and jump 142
A-14 A template video for each class in the video classification application database: left-to-right, circular, and hop 143

B-1 Karate punch query video with the *Cum* hypervolume's dimension = $11 \times 12 \times 12 \times 7$ 147
B-2 Warped target video using Multiscale DTSW with the *Cum* hypervolume's dimension = $11 \times 12 \times 12 \times 7$ 147
B-3 Warped target video using Piece-wise Multiscale DTSW with the *Cum* hypervolume's dimension = $11 \times 12 \times 12 \times 7$ 148

List of Tables

3.1 Comparison of execution time of DTSW and Multiscale DTSW for various relaxation radii on various sets of video, running on an IBM Pentium M laptop (1.3 GHz processor with 768 MB memory) 69

3.2 Comparison of normalized execution time of Multiscale DTSW for various relaxation radii on various sets of video, running on an IBM Pentium M laptop (1.3 GHz processor with 768 MB memory) 70

3.3 Comparison of error of Multiscale DTSW for various relaxation radii on various sets of video, running on an IBM Pentium M laptop (1.3 GHz processor with 768 MB memory) 73

3.4 Comparison of execution time of DTSW and Multiscale DTSW for various relaxation radii in comparing the karate punch and horse racing videos running on a PC with 3.4 GHz processor and 1 GB memory . 73

3.5 Comparison of normalized execution time of DTSW and Multiscale DTSW for various relaxation radii in comparing the karate punch and horse racing videos running on a PC with 3.4 GHz processor and 1 GB memory 75

3.6 Comparison of error of DTSW and Multiscale DTSW for various relaxation radii in comparing the karate punch and horse racing videos running on a PC with 3.4 GHz processor and 1 GB memory 75

4.1 The normalized execution time of DTSWEF in comparing the karate punch, heart valve, and palm opening and closing videos with the variance = {100%, 90%, 80%, 70%} 82

4.2 The normalized error of DTSWEF in comparing the karate punch, heart valve, and palm opening and closing videos with the variance = {100%, 90%, 80%, 70%} 82

4.3 The normalized execution time and the normalized error of DTSW, Multiscale DTSW, and Multiscale DTSWEF in comparing the karate punch, heart valve, and palm opening and closing videos with the constraint that the normalized error must be less than 10% 86

4.4 Comparison of execution time of Multiscale DTSW and Multiscale DTSW with Control Points running on a Pentium M laptop with 1.3 GHz processor and 768 MB memory in comparing the karate punch videos 89

4.5 Comparison of execution time of DTSW, Multiscale DTSW, and Multiscale DTSW with Level Jump in comparing the karate punch videos running on a Pentium M laptop with 1.3 GHz processor and 768 MB memory 93

5.1 The accuracy of the video classification application on the training and unknown sets for different motion scalar combinations 101

5.2 The accuracy of the video classification application on the training and unknown sets for various settings of the normalized motion scalar combination 105

5.3 The accuracy of the video classification application on the training and unknown sets for various sets of resolution with $u_N{}^2$ as the motion scalar combination 106

5.4 The accuracy of the video classification application on the training and unknown sets for various sets of resolution with $\mathbf{u}$ as the motion scalar combination 109

5.5 The accuracy of the video classification application on the training set for various values of b_x and b_y 110

5.6 The accuracy of the Multiscale-DTSWEF-based video classification application on the training set with various settings for the variance . . 111

5.7 The normalized execution time and percentage error of the video classification application based on DTSW, Multiscale DTSW, and Multiscale DTSWEF on the unknown set 111

5.8 The accuracy of the Multiscale-DTSWEF-based video classification application on the training set (left-to-right, circular, and hop sets) for various settings for the variance 112

5.9 The normalized execution time and accuracy of the video classification application based on DTSW and Multiscale DTSW on the unknown set in classifying left-to-right, circular, and hop pointing actions . . . 113

5.10 The accuracy of the video classification application on the unknown set for various scale differences between the template and unknown videos 113

6.1 The accuracy in the time dimension of the DSW-based video detection application on 40 target cases with various temporal detection offsets. The optical flow component used was $\mathbf{u}$. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). 120

6.2 The accuracy in the space dimension of the DSW-based video detection application on 40 target cases with various spatial detection offsets. The optical flow component used was $\mathbf{u}$. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). 121

6.3 The accuracy in the time dimension of the Multiscale-DTSW-based video detection application on 40 target cases with various temporal detection offsets. The optical flow component used was $\mathbf{u}$. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). 121

6.4 The accuracy in the space dimension of the Multiscale-DTSW-based video detection application on 40 target cases with various spatial detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). 122

6.5 The accuracy in the time dimension of the DSW-and-Multiscale-DTSW-based video detection application on 200 target cases with various temporal detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). 123

6.6 The accuracy in the space dimension of the DSW-and-Multiscale-DTSW-based video detection application on 200 target cases with various spatial detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension).123

6.7 The accuracy in the time dimension of the DSW-based video detection application on 200 target cases with various temporal detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). 124

6.8 The accuracy in the space dimension of the DSW-based video detection application on 200 target cases with various spatial detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). 124

6.9 The average accuracy in the time dimension of the video detection application on 40 target cases for various scale differences between the query and target videos. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). The method used was the combination of DSW and Multiscale DTSW. 125

6.10 The average accuracy in the space dimension of the video detection application on 40 target cases for various scale differences between the query and target videos. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). The method used was the combination of DSW and Multiscale DTSW. 125

B.1 Comparison of the similarity between the query video and the warped target video found using Multiscale DTSW and Piece-wise Multiscale DTSW. The similarity between two videos is measured by the absolute difference between the two videos. The absolute difference between two videos is defined as the summation of the absolute difference in the grayscale value of each pixel in each frame of the two videos. The higher the absolute difference is, the more dissimilar the two videos are. The execution time for each experiment running on an IBM Pentium M laptop (1.3 GHz processor with 768MB memory) is also included. . 146

Chapter 1

Introduction

1.1 Motivation

There are many applications that require a quantified comparison between two videos in a short time. Below are five categories of such applications.

1.1.1 Video Query Application

In most video query applications for example YouTube and Google video, we can find a video in a video database by searching for keywords. Keywords are generated from audio tracks or manually entered. Queries are limited to these words. Often we are interested in the videos visual contents.

Video content query is one solution to this problem. Such a video query application uses a video clip as an example and returns the similar video clips from the video database. In order to be practical, the video query application must return the similar videos within a few seconds. We need to have an algorithm that can rapidly compare two videos.

1.1.2 Video Detection Application

In video detection applications, we search for a query video inside of a temporally and spatially larger target video. Video detection applications can be online or off-line.

Online video detection application means that the application is trying to find a query video inside a real-time captured video. One of the applications is a video-based autonomous monitoring system for failure detection in the manufacturing industry. The system compares a real-time video clip of one cycle of a manufacturing process with a target video of the ideal one cycle of the manufacturing process. If a deviation to the ideal manufacturing process is detected based on the video clips' dissimilarity, the system will shut down the manufacturing machine and report a system failure. Early failure detection is important in the manufacturing industry to prevent the failure from propagating to other parts of the manufacturing process and causing more serious damage. In addition, because the damage is isolated, we will know exactly which part of the manufacturing machinery that causes the problem.

On the other hand, for off-line video detection applications, the target video is recorded. Hence, these applications do not really need to enforce a time constraint in obtaining the detection result. For example, we are given a soccer match video and we are only interested in the goals. So we run a video detection application to find all occurrences of goal. Maybe we do not really mind if the result comes out five to ten minutes after we run the application. However, we do not want to wait for hours to retrieve the goals. Therefore, in online or off-line video detection applications, there is a need for a fast algorithm in comparing two videos.

1.1.3 Video Classification Application

In video classification applications, we find if an unknown video belongs to one of several classes or categories of video. For example, we record several athletic events, and then we want the videos to be classified into running, throwing, and jumping. One or several videos for each class contain the represented activity – this is the training set. We compare an unknown video to the training set to determine to which class it belongs.

1.1.4 Combination of Video Classification and Detection Application

There are some applications that are a combination of video classification and video detection. For instance, an application that interprets videos of person performing sign language for deaf people. The application has a "dictionary" that consists of all videos showing every word in the sign language. Firstly, the application must be able to detect which subsequence of frames of the input video that represents a word and which part of the video that does not represent any sign language's words. Secondly, the application needs to search for the best match between the detected subsequence of frames and the application's sign language dictionary to interpret the detected sign language's word into a verbal word.

This application is computationally expensive because it needs to perform hundreds or thousands of video clips comparison. Therefore, to reduce its complexity, this application really needs an inexpensive video comparison algorithm.

1.1.5 Video Comparison Based Judging

Think of a sports competition where the winner of the competition is decided by a panel of judges based on their decisions on how well the acts performed by the athlete. One example is a diving competition. Since the scores at the diving competition are only based on human's evaluation, the scores may not be as objective as it could be. If we can develop an application that can give a comparison between the movement performed by the athlete and the ideal movement, the objectivity of the scoring system of the competition can be improved. Although this application is not meant to replace the panel of judges, it can help the judges on their decision. Due to the nature of this application, the comparison of two videos should take less than a few minutes. If the application takes longer than few minutes, it will be useless since the judges must give their scores within few minutes after the athlete has performed.

In summary, there are a lot of applications that requires a fast video comparison

algorithm.

1.2 Background

Dynamic Time Warping (DTW) is a technique to calculate the similarities between two time series. DTW aligns two time series by compressing or stretching the time axis. DTW is widely used in various research fields: gesture recognition [6], speech recognition [19], data mining [2, 13, 29], manufacturing [7], medicine [5], ECG pattern matching [5, 27], robotics [18, 24], biometric data alignment [6, 11, 15, 16], and polymerization synchronization [11].

An extension of DTW, Dynamic Time and Space Warping (DTSW), has been developed by Anthony [1] to find the similarities between two video clips. The algorithm optimally warps time and shifts space in order to align the query video with the target video. The output of DTSW is an optimal warp path in the time and space for optimal video alignment.

In [1], Anthony demonstrates the applicability of DTSW to industrial wear monitoring, failure prediction, and assembly-line feedback control. The DTSW technique can also be applied to video detection and classification applications.

1.3 Problem Statement

As mentioned in the motivation section, there are numerous applications that require an algorithm that can compare two videos and determine their similarities in a short time.

Although DTSW is good in comparing two video clips, it is computationally expensive. DTW, which works on a single value series, needs quadratic time and space complexity ($O(N^2)$). DTSW, which is an extension of DTW and works on a 2-D value series, needs $O(N^4)$ time and space complexity.

1.4 Contributions

In this thesis, we introduce Multiscale DTSW, a modification of DTSW that has linear time and space complexity ($O(N)$) without greatly reducing the accuracy of the comparison results.

Extensions of Multiscale DTSW utilize additional information about the two input videos to further reduce the computational cost.

A video classification application that is based on Multiscale DTSW is developed. It accomplishes the same classification result as a DTSW-based video classification application with lesser execution time.

A video detection application that combines Dynamic Space Warping (DSW) and Multiscale DTSW techniques is developed. It detects the temporal and spatial location of a query video inside a target video in a fast and accurate manner.

1.5 Literature Review

Dynamic Time Warping (DTW) algorithm [8, 12, 17, 21] optimally aligns two time series that have variation along the time axis. B. W. Anthony [1] extends the DTW algorithm to align two videos and include the variation along the spatial axis in addition to the variation along the time axis. The algorithm is called Dynamic Time and Space Warping (DTSW). DTSW aims to find the optimal way to warp the time and shift the space of the query video such that it is as similar as possible to the target video. DTW and DTSW are implemented using the dynamic programming technique.

As discussed in [1], the advantages of DTSW compared to existing appearance-based techniques include:

- DTSW can be used to determine detailed similarities between two videos as a function of time and space.
- DTSW is structured such that it can be implemented using parallel processing to increase the rate of operation.

- DTSW uses motion, pixels, or any other volumetric data that is application appropriate.
- DTSW is appropriate for subject matter that is amorphous and flexible.
- DTSW can be used for video comparison or alignment, video event detection, and video classification applications.

DTSW has a parallel structure that makes it fast enough to be relevant to real-time industrial applications. Parallel processor and computation is often expensive. This motivates us to research strategies to reduce the computational cost of DTSW without significant degradation in accuracy.

Because DTSW is an extension of DTW, many techniques built to reduce the complexity of DTW can be applied to DTSW to reduce the complexity of DTSW.

The first technique is to reduce the computations of the DTW cost matrix by selecting particular region in the DTW cost matrix as the search region. In other words, we enforce a constraint in the DTW optimization problem. The most commonly used constraints are the Sakoe-Chuba Band [22] and the Itakura Parallelogram [9]. The width of the search region of the DTW cost matrix (or window) is specified by a parameter. The optimal warp path is found within the search region. Therefore, the optimal warp path found by this technique may not be globally optimal if the global optimal warp path does not lie within the search region. This speed-up technique can be used only if we expect the optimal warp path to be nearly straight diagonal path or to lie inside the search region.

A second technique to reduce computational cost is to run DTW on downsampled inputs [4, 13]. A warp path is found at coarse time resolution and then the warp path is projected to the original resolution without any further refinement. The projected warp path is the final solution to the DTW problem. Because this method ignores the local variations that could occur in between the resolutions, the warp path found by this method may be far from optimal.

The third technique is to apply a multilevel approach that is similar to the multilevel approach used for graph bisection[10]. The objective of graph bisection is to divide a graph into two partitions with a constraint that the sum of the weights of the edges connecting the two partitions is as small as possible. For the graph bisection problems, there exists some efficient algorithms for solving small graphs problems. However, the algorithms are not efficient for large graphs. In the multilevel approach, the large graph to be partitioned is downsampled into a small graph. The solution for the bisection of the small graph can be computed efficiently using the existing algorithm. The solution is then projected to a slightly larger graph, and the projected solution in the larger graph is then refined. Refinement to the partial (non optimal) solution can be found efficiently. The process of projecting the solution to a slightly larger graph and refining the solution is repeated until the solution to the original-size graph is found.

In this research, we apply a multilevel or multiscale approach to DTSW. The reason of chosing multiscale approach to reduce the complexity of DTSW is that FastDTW algorithm [23] developed by S. Salvador and P. Chan has successfully applied this multiscale approach to DTW and proven that FastDTW can achieve almost optimal results with linear complexity. Another reason to use multiscale approach rather than other approaches is that we can derive scale information regarding the movement or action in the input videos based on the solution at different levels of resolution. Based on the solution for the coarsest resolution, we can compare the low frequency movement between the two input videos. Similarly, based on the solution for the finest resolution, we can compare the high frequency movement between the two input videos. This will be useful if we want to classify a video to two or more different classes based on its low or high frequency motion content.

1.6 Document Outline

The rest of this document is organized as follows. In Chapter 2 we review DTW and DTSW. In Chapter 3 we develop the Multiscale DTSW algorithm and explain

its computational complexity. We include experimental results. In Chapter 4 we explore several extensions of Multiscale DTSW that reduce the complexity of Multiscale DTSW. In Chapter 5 we develop a video classification application where Multiscale DTSW is shown to be very efficient. In Chapter 6 we build a video detection application using a combination of DSW and Multiscale DTSW methods. In Chapter 7 we summarize the contribution of this research and suggest future research directions.

Chapter 2

Related Algorithms

2.1 DTW

Dynamic Time Warping (DTW) [14, 23] is a technique that compares two time series by finding the optimal alignment between the two time series. DTW compresses or stretches one of the time series along its time dimension in order to match it with the other time series.

2.1.1 DTW Algorithm

The problem statement of DTW is defined as follows . There are two time series, C of length I, and Q of length J, where

$$C = c_1, c_2, ...c_i, ..., c_I, \tag{2.1}$$

and

$$Q = q_1, q_2, ...q_j, ..., q_J. \tag{2.2}$$

The objective of DTW is to find the optimal warp path. The optimal warp path is the path that has the minimum cost of aligning the two time series or the minimum

warp cost. The notation of a warp path is

$$W = w_1, w_2, ...w_k, ..., w_K, \tag{2.3}$$

where max(I,J) $\leq K \leq (I + J)$. Variable K is the length of the optimal warp path, and the k^{th} element of the optimal warp path is

$$w_k = (j_k, i_k), \tag{2.4}$$

where j_k is an index in time series Q, and i_k is an index in time series C.

A warp path is a sequence of 2-D indices that must satisfy:

- **Boundary condition**: w_1 must be equal to (1,1) and w_k must be equal to (J,I). These conditions restrict a warp path to start at the beginning of both time series and finish at the end of both time series.

- **Continuity condition**: If $w_k = (a, b)$, then $w_{k+1} = (a', b')$, where $a - a' \leq 1$, and $b - b' \leq 1$. This requirement restricts a warp path from moving to any but adjacent cells.

- **Monotonicity condition**: If $w_k = (a, b)$, then $w_{k+1} = (a', b')$, where $a - a' \geq 0$, and $b - b' \geq 0$. In other words, the monotonicity condition means that the 2-D indices in a warp path have to be monotonically increasing.

The first step in DTW is to construct a J-by-I distance matrix (D) where the (j^{th},i^{th}) element of the matrix contains the distance between q_j and c_i. The distance between q_j and c_i is defined as $D(j, i) = (q_j - c_i)^2$.

The next step is to build a J-by-I cumulative matrix *(Cum)*. Each element of the Cum matrix – $Cum(j, i)$ – is a summation of the distance between q_j and c_i – $D(j, i)$ – and the cumulative distance of its adjacent elements:

$$Cum(j, i) = D(j, i) + min\{Cum(j - 1, i), Cum(j, i - 1), Cum(j - 1, i - 1)\}. \tag{2.5}$$

There are only three adjacent elements in Equation 2.5 due to the continuity and monotonicity restrictions of a warp path.

The final step in DTW is to find the optimal warp path. There are exponentially many possible warp paths. However, the optimal warp path is the path that has the minimum warp cost:

$$d^*_{DTW}(Q,C) = min\frac{\sqrt{\sum_{k=1}^{K} D(w_k(j), w_k(i))}}{K}. \tag{2.6}$$

The K in the denominator is used to normalize the warp cost as different warp paths may have different lengths. Due to the boundary condition of a warp path, the two ends of the optimal warp path must be $(1,1)$ and (J,I).

DTW starts a process of searching for the optimal warp path at cell (J,I). DTW then selects which path it should move to by comparing the value of the adjacent cells in the *Cum* matrix. The adjacent cells are the cells to the left, down, and diagonally to the down-left of the current cell. Whichever of these three adjacent cells that has the smallest value in the *Cum* matrix is added to the optimal warp path found so far, and the searching moves to the newly added cell. The steps of comparing the values of the adjacent cells in the *Cum* matrix, adding the cell with the smallest value to the optimal warp path, and moving to the newly added cell, are repeated until the searching reaches cell (1,1). The optimal warp path is the set of indices of the cells that have been passed through by the searching process from (J,I) to (1,1).

2.1.2 Complexity of DTW

Assume that the length of time series C (I) and the length of time series Q (J) are both equal to N; then the time and space complexity of DTW are analyzed as follows.

- Time complexity

  DTW has two major computations:

  - *Building matrices*

    DTW needs to compute a distance and cumulative matrix (D and *Cum*).

Both matrices are of size $I \times J = N \times N$. Therefore, the cost of building the distance and cumulative matrix is equal to $2N^2$.

- *Searching for the optimal warp path*

  In the worst case (Equation 2.3), the optimal warp path is of length $I+J = N+N$. Thus, the cost of finding the optimal warp path is equal to $2N$.

In summary, the total time complexity of DTW is equal to $2N^2 + 2N$.

- Space complexity

  DTW needs to store a distance matrix (D) and a cumulative matrix (Cum). Each matrix is of size $I \times J = N \times N$. Therefore, to store both matrices, DTW needs $2N^2$ of space. Hence, the space complexity of DTW is equal to $2N^2$.

In conclusion, DTW has quadratic time and space complexity ($O(N^2)$).

2.1.3 FastDTW

DTW algorithm is only suitable for comparing short time series because of its quadratic time and space complexity. The computational cost of DTW algorithm in comparing long time series is enormous. Therefore, there is a need for methods that can speed up the DTW algorithm. One of the methods is called FastDTW [23], which was developed by S. Salvador and P. Chan.

FastDTW applies multiscale approach to DTW to reduce its complexity to linear complexity. Firstly, FastDTW downsamples the input time series to the coarsest resolution. Secondly, FastDTW applies the DTW algorithm to the coarsest resolution time series to obtain a solution for the coarsest resolution. Subsequently, FastDTW projects the solution for the coarsest resolution to finer resolution. FastDTW then does some refinement to the projected solution to improve the accuracy of the solution. FastDTW then repeatedly projects the solution to finer resolution and refines the solution until the solution for the finest resolution is obtained. The linear time and space complexity of FastDTW have been proven both theoretically and empirically in [23].

Although FastDTW algorithm is not guaranteed to find the optimal solution, based on the experiment carried out by S. Salvador and P. Chan, the accuracy of FastDTW's solution as compared to the solution of DTW algorithm is satisfactory.

2.2 DTSW

Dynamic Time and Space Warping (DTSW) [1] is an extension of DTW and was developed by B. W. Anthony (2006) to determine a detailed comparison in the time and space dimensions between a query and target video. DTSW compares two videos – a query and target video– by finding the optimal path to align the query video in the time and space dimensions of the target video.

In DTSW, a video that is spatially larger than the other is the target video, and a video that is spatially smaller than the other is the query video. The target video can be temporally shorter or longer than the query video.

2.2.1 DTSW Algorithm

The problem statement of DTSW is defined as follows. There are two videos: a query video, Q, with a length of J frames and a size of $[N_Q \times M_Q]$ for each frame; and a target video, C, with a length of I frames and a size of $[N_C \times M_C]$ for each frame. Q has smaller spatial size; $N_Q \leq N_C$ and $M_Q \leq M_C$. The objective of DTSW is to find the optimal warp path. The optimal warp path is the path that has the minimum cost of aligning the two videos or the minimum warp cost. The notation of the optimal warp path is

$$W = w_1, w_2, ...w_k, ..., w_K, \tag{2.7}$$

where $\max(I,J) \leq K \leq (I + J)$. Variable K is the length of the optimal warp path, and the k^{th} element of the optimal warp path is

$$w_k = (j_k, i_k, x_k, y_k), \tag{2.8}$$

where j_k is an index in the query video Q, i_k is an index in target video C, and $\{x_k,y_k\}$ is a possible location of aligning frame Q_{j_k} to frame C_{i_k}.

A warp path is a sequence of 4-D indices that must satisfy:

- **Spatial continuity**: There is a bound in the spatial change from frame to frame.
- **Spatial drift**: If two or more frames of the query video are matched to a frame in the target video, then the frame-to-frame matching must occur at the same spatial location.
- **Temporal continuity**: This is the the continuity condition of DTW.
- **Bi-Temporal casuality**: The sequence in both videos cannot be out of the time sequential order.
- **Boundary condition**: The first frame of the target video must be matched to the first frame of the query video. Likewise, the last frame of the target video must be matched to the last frame of the query video.

The first step in DTSW is to calculate the elemental distance between every frame of the query video and every possible subregion of each frame of the target video. DTSW then places the results in an *Elemental Distance Hypervolume, D*. The elemental distance in DTSW is defined as the similarity measurement between a frame of the query video and a similar size region of a frame of the target video. This is because we assume that we can only calculate the similarity measurement between regions that have the same size. Then, each frame of the query video can be positioned in $X = N_C - N_Q$ different spatial locations horizontally, $Y = M_C - M_Q$ different spatial locations vertically, and I different temporal locations. Hence the size of the *Elemental Distance Hypervolume* (D) is $|I \times J \times X \times Y|$.

The next step is to compute a *Cumulative Distance Hypervolume, Cum*, with a size of $|I \times J \times X \times Y|$. Each cell of the Cum hypervolume – $Cum(j,i,x,y)$ – is the summation of the elemental distance in that cell – $D(j,i,x,y)$ – and the cumulative

distance of the adjacent cells:

$$Cum(j,i,x,y) = D(j,i,x,y) + \begin{cases} Cum(j-1,i,x,y) \\ Cum(j,i-1,x-b_x,y-b_y) \\ Cum(j-1,i-1,x-b_x,y-b_y) \end{cases} \tag{2.9}$$

The adjacent cells defined in Equation 2.9 are based on the spatial continuity, spatial drift, temporal continuity, and bi-temporal casuality restrictions of a warp path. Variables b_x and b_y are the maximum allowable changes in the x and y dimensions between a frame and its adjacent frame.

The final step in DTSW is to find the optimal warp path. There are exponentially many possible warp paths. However, the optimal warp path is the path that has the minimum warp cost:

$$d^*_{DTSW}(Q,C) = min\frac{\sum_{k=1}^{K} D(w_k(j),w_k(i),w_k(x),w_k(y))}{K}. \tag{2.10}$$

The K in the denominator is used to normalize the warp cost as different warp paths may have different lengths. Due to the boundary condition of a warp path, the two ends of the optimal warp path must be $(1,1,x_1,y_1)$ and (J,I,x_K,y_K).

DTSW starts a process of searching for the optimal path by computing the minimum of {$Cum(J,I,1,1)$, $Cum(J,I,1,2)$, ..., $Cum(J,I,1,Y)$, ..., $Cum(J,I,2,1)$, ..., $Cum(J,I,2,Y)$, ..., $Cum(J,I,X,Y)$}. The (J,I,x,y) index of the minimum value will be $(J,I,w_K(x),w_K(y))$. DTSW then selects which path it should move to by comparing the value of the adjacent cells of the current cell in the Cum matrix. The adjacent cells are the cells in {$Cum(J-1,\ I,\ w_K(x),\ w_K(y))$, $Cum(J,\ I-1,\ w_K(x)$ - b_x, $w_K(y)$ - b_y), $Cum(J-1,\ I-1,\ w_K(x)$ - b_x, $w_K(y)$ - b_y)}. Whichever of these adjacent cells that has the smallest value in the Cum matrix is added to the optimal warp path found so far, and the searching moves to the newly added cell. The steps of comparing the value of the adjacent cells in the Cum matrix, adding the cell with the smallest value to the optimal warp path, and moving to the newly added

cell are repeated until the searching reaches cell $(1, 1, x_1, y_1)$. There is no restriction for the value of x_1 and y_1. The optimal warp path is the set of indices of the cells that have been passed through by the searching process from $(J, I, w_K(x), w_K(y))$ to $(1, 1, x_1, y_1)$.

2.2.2 Complexity of DTSW

Assume that I, J, X, and Y in the D and Cum hypervolumes are all equal to N; then the time and space complexity of DTSW are analyzed as follows.

- Time complexity

  DTSW has two major computations:

  - *Building hypervolumes*

    DTSW needs to compute an *Elemental Distance* and *Cumulative Distance* hypervolumes (D and Cum). Both hypervolumes are of size $I \times J \times X \times Y = N \times N \times N \times N$. Therefore, the cost of building the *Elemental Distance* and the *Cumulative Distance* hypervolumes $= 2N^4$.

  - *Searching for the optimal warp path*

    In the worst case (Equation 2.7), the optimal warp path is of length $I + J = N + N$. For each element in the time dimension of the warp path, DTSW needs to compare $b_x \times b_y$ elements in the space dimension. Thus, the cost of finding the optimal warp path is $2N \times b_x \times b_y$.

  In summary, the total time complexity of DTSW is

  $$2N^4 + (2N \times b_x \times b_y). \tag{2.11}$$

- Space complexity

  DTSW needs to store an *Elemental Distance* and *Cumulative Distance* hypervolume. Each hypervolume is of size $I \times J \times X \times Y = N \times N \times N \times N = N^4$. Therefore, to store both hypervolumes, DTSW needs $2N^4$ of space. Hence, the space complexity of DTSW is $2N^4$.

In conclusion, DTSW has $O(N^4)$ time and space complexity.

Chapter 3

Multiscale DTSW

Multiscale DTSW is an extension of DTSW that uses a multiscale approach to reduce complexity without greatly reducing accuracy.

In Multiscale DTSW, the DTSW algorithm is applied to coarse resolution (down-sampled in time and space) input videos. Multiscale DTSW then projects the solution from coarse resolution to finer resolution. Based on the projected solution, a solution for the finer resolution is refined. Multiscale DTSW then repeatedly projects the solution from the current resolution to finer resolution until the desired resolution is reached.

Figure 3-1 shows the diagram of DTSW for comparing two input videos with a length of 8 frames and 6 frames. The *Elemental Distance* hypervolume is fully filled and computed. Based on the filled cells in the *Elemental Distance* hypervolume, the optimal warp path is computed. Multiscale DTSW with two levels of resolution applied to the same input videos is shown in Figure 3-2. We call the two levels of resolution as coarse resolution and fine resolution. In the first step, the input videos are downsampled in the time and space dimensions. In the second step, the *Elemental Distance* hypervolume for the coarse resolution is fully filled and computed. In the third step, the optimal warp path for coarse resolution is obtained. In the fourth step, the optimal warp path for coarse resolution is projected to the *Elemental Distance* hypervolume for fine resolution. In the fifth step, the *Elemental Distance* hypervolume for fine resolution is partially filled on the projected cells. In the final step, the optimal

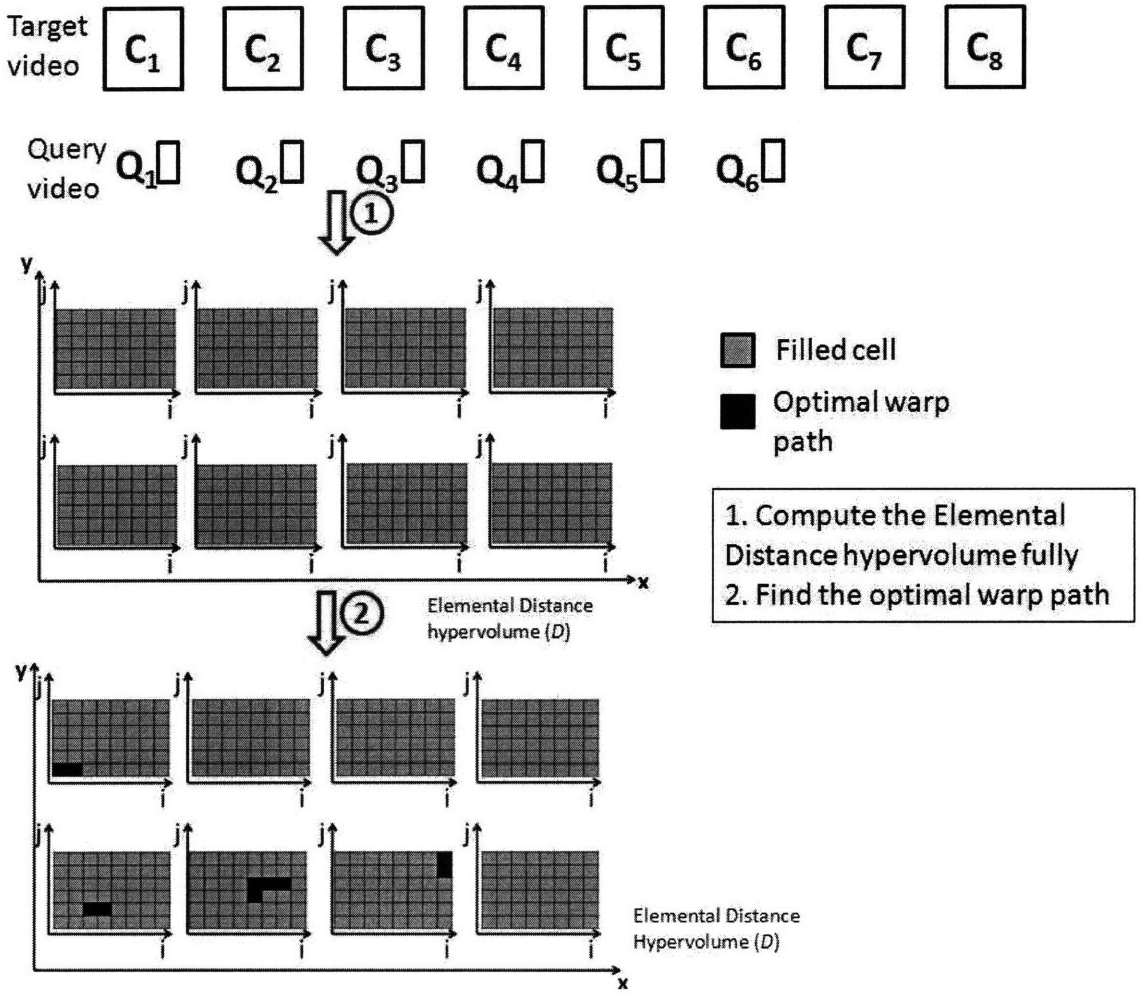

Figure 3-1: Diagram of DTSW

warp path for fine resolution is computed based on the filled cells in the *Elemental Distance* hypervolume. In this way Multiscale DTSW computes the path constraints conceptually similar to the Sakoe-Chuba Band [22] and the Itakura Parallelogram [9] but based on the input data.

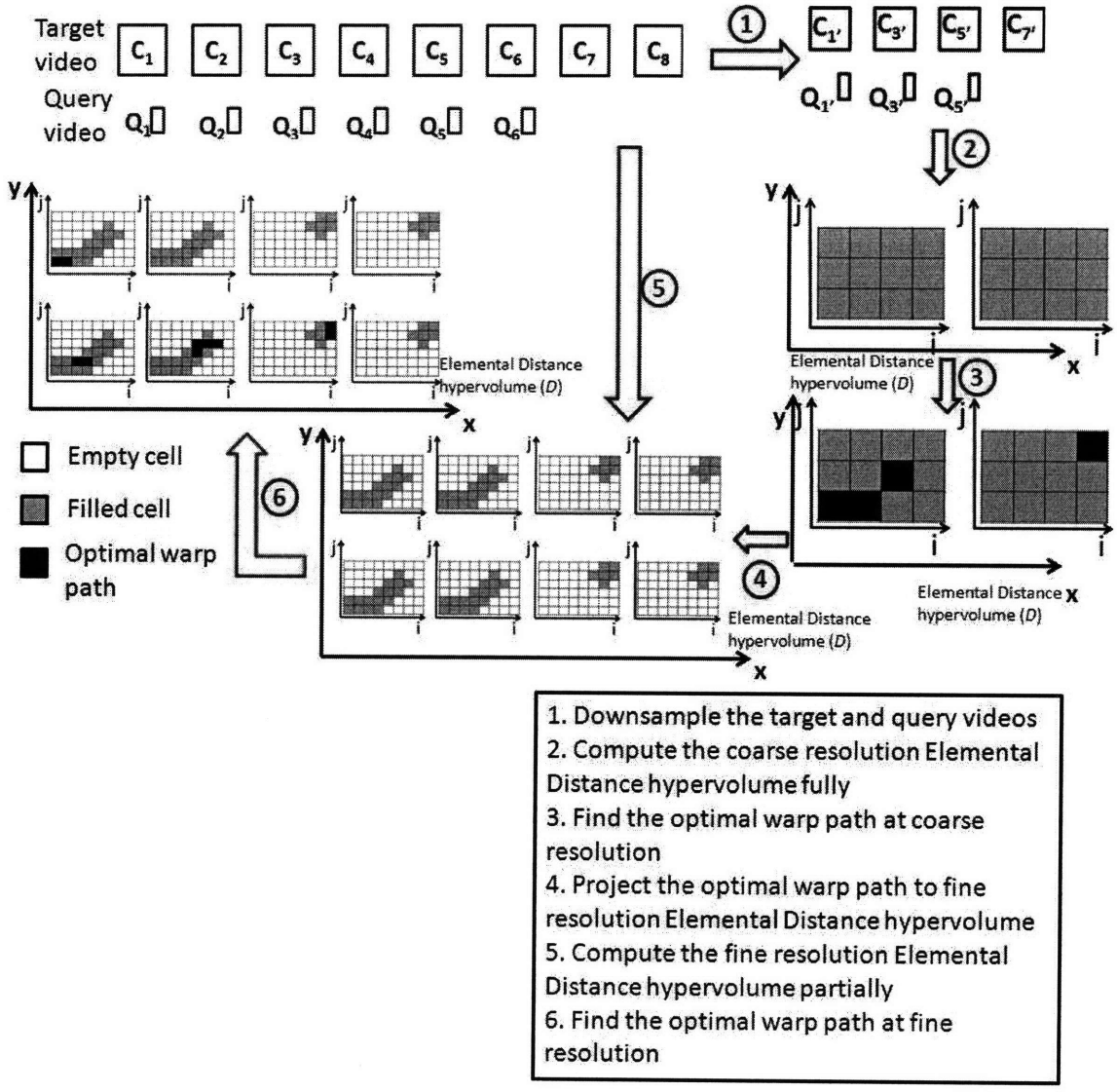

Figure 3-2: Diagram of Multiscale DTSW with optimal level = 2, $r_s = 0$, $r_t = 0$

By comparing the two diagrams, we expect that the total computation of Multiscale DTSW to be less than of DTSW due to a great computational saving for the *Elemental Distance* hypervolume computation at fine resolution. The details about the total computation of Multiscale DTSW can be found in Section 3.2.

The next section discusses the Multiscale DTSW algorithm in more details.

3.1 Multiscale DSTW Algorithm

Algorithm 3.1 summarizes the Multiscale DTSW algorithm in a pseudocode. Figure 3-3 depicts the flowchart of the Multiscale DTSW algorithm.

First, Multiscale DTSW computes the optimal level and radii based on the size of the input videos and a relaxation radius that a user wants. The optimal level is the number of resolution levels that minimizes the total computation of Multiscale DTSW. The optimal level of three means that there are three levels in Multiscale DTSW. At level 1, the full DTSW is applied. For level 2 and level 3, the optimal warp path is obtained by refining the projected warp path from lower level. At level 1, the input videos are downsampled twice in time and space dimensions. At level 3, there is no downsampling on the input videos. A relaxation radius is a parameter that determines how similar the solution of Multiscale DTSW to the solution of DTSW. Based on relaxation radius and the size of the input videos, Multiscale DTSW computes radii: r_s and r_t. Variable r_s is a radius that determines how big the search region in space dimension that Multiscale DTSW searches for the optimal warp path around the projected warp path. Variable r_t is a radius that determines how big the search region in time dimension that Multiscale DTSW searches for the optimal warp path around the projected warp path. The bigger the relaxation radius is, the more similar the solution of Multiscale DTSW to the solution of DTSW. To obtain a similar solution to DTSW, the search region for the optimal warp path in Multiscale DTSW must be big. Note that the search region of DTSW is the full hypervolume while the search region of Multiscale DTSW is a subset of the hypervolume. Hence, for big relaxation radius, the radii (r_t and r_s) are also big.

Second, the input videos are downsampled in time and space dimensions according to the computed optimal level. Third, the full DTSW is applied to the coarsest resolution input videos. For the subsequent levels (if the optimal level is not equal to one), Multiscale DTSW projects the optimal warp path found at the current resolu-

Algorithm 3.1 Multiscale DTSW

Input:
C - a target video
Q - a query video
rr - relaxation radius
b_x - restriction in horizontal change in the space dimension
b_y - restriction in vertical change in the space dimension

Output:
WarpPath - the optimal warp path
S - the similarity measurement between the target and query videos

```
[optimal_level, radii] = Compute_Optimal_Level(C,Q,rr)

level = 1
[C_downsample, Q_downsample] = Downsample_video(C,Q,optimal_level,
level)
[WarpPath, S] = FullDTSW(C_downsample, Q_downsample, b_x, b_y)
```

for *level* = 2 to *optimal_level* **do**

```
  prev_S = S
  [C_downsample, Q_downsample] = Downsample_video(C,Q,optimal_level,
  level)
  [Predicted_WarpPath] = Upsample_WarpPath(WarpPath)

  [WarpPath,S] = PartialDTSW(C_downsample, Q_downsample,
  Predicted_WarpPath, radii, b_x, b_y)
```

if Compare(prev_S,S) == true **then**
  break;
end if

level $\leftarrow$ *level* + 1

end for

return WarpPath, S

tion to finer resolution *Elemental Distance* and *Cumulative Distance* hypervolumes. Multiscale DTSW then fills in the finer resolution hypervolumes only at the projected path or cells. The optimal warp path for finer resolution is recomputed based on the filled cells in the hypervolumes. The process of projecting the optimal warp path to finer resolution and recomputing the optimal warp path is repeated until Multiscale DTSW reaches the optimal level.

The levels in Multiscale DTSW are numbered from one to the optimal level. The level refers to the resolution level in Multiscale DTSW. If the optimal level is five, the levels in Multiscale DTSW are numbered as 1, 2, 3, 4, and 5. At level 1, the input videos are downsampled to the coarsest resolution. At level 4, both input videos are downsampled once. And at level 5, there is no downsampling on the input videos. Level 5 is for the finest resolution.

The downsampling multiplier of the input videos in the time and space dimensions may not necessarily be two. It depends on the speed and the spatial change of the action in the videos. If the speed of the action in the videos from frame to frame is slow, then Multiscale DTSW will downsample the videos in the time dimension by a multiplier of four or eight at each level. The downsampling multiplier in the space dimension is independent to the downsampling multiplier in the time dimension. The downsampling multiplier in the space dimension depends on how far the spatial change of the action in the videos from frame to frame. If the spatial change is big, then Multiscale DTSW will downsample the videos in the space dimension by a multiplier of four or eight at each level. In this research, the downsampling multiplier in the time and space dimensions is two. All equations in this thesis are based on the assumption that the downsampling multiplier in the time and space dimensions is two.

Unlike FastDTW [23], that performs the multiscale approach until the coarsest possible resolution, Multiscale DTSW performs the multiscale approach only up to the optimal level (the number of resolution levels that minimizes the total computation of Multiscale DTSW). More explanation about the optimal level can be found in Section 3.3.1.

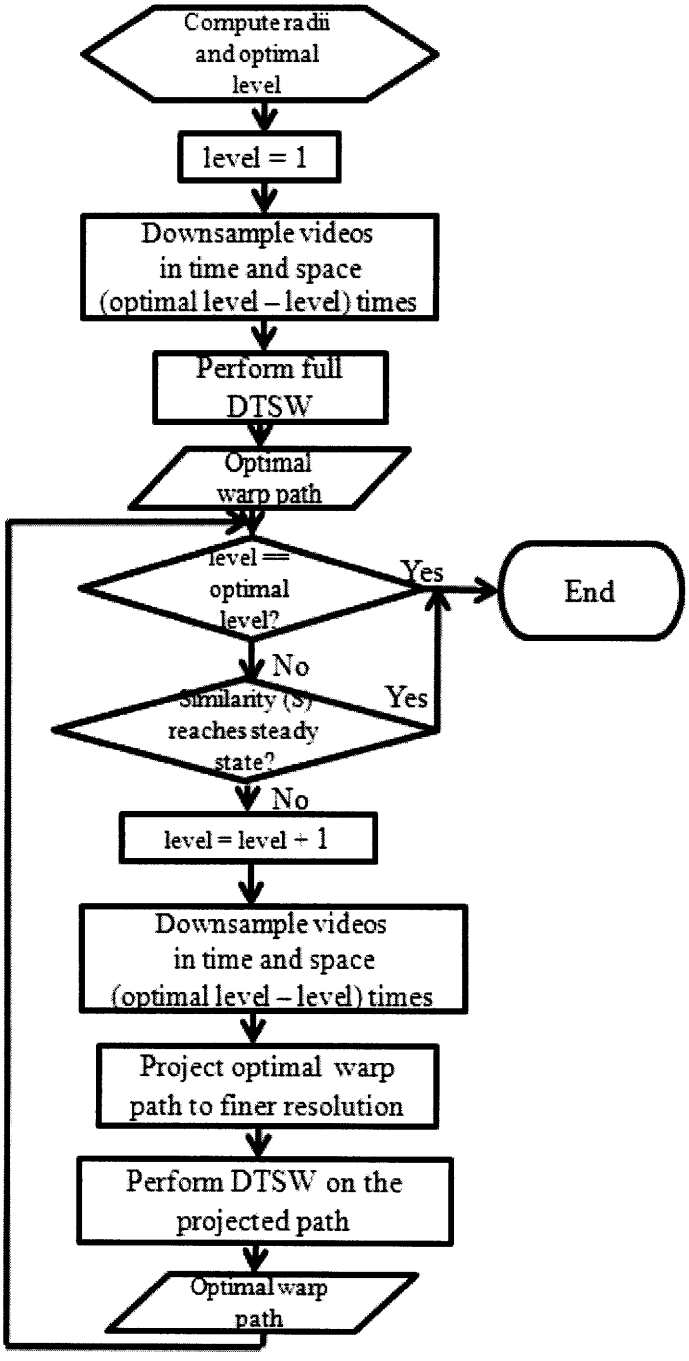

Figure 3-3: Flowchart of Multiscale DTSW

The similarity measurement (S) between two videos (a query and target video) with a length of I and J frames, respectively, is defined by

$$S = \frac{Cum(J, I, x_K, y_K)}{I + J}. \tag{3.1}$$

The smaller the value of S is, the more similar the two videos are. DTSW and Multiscale DTSW are used to evaluate the similarity between the two input videos.

Figures 3-4 to 3-7 shows the S at each level of Multiscale DTSW's execution in comparing the video of karate punch moves, horse racing, heart valve opening and closing, and palm opening and closing. These videos can be found in Appendix A. As can be observed from the figures, as the level of Multiscale DTSW increases (as downsampling decereases), the value of S converges. The higher the level is, the less number of downsampling is applied to the input videos. The highest level is for the finest resolution and level 1 is for the coarsest resolution. The level at which the value of S converges varies from one set of videos to another set of videos. Therefore, we cannot predict or preset at which level the value of S will converge.

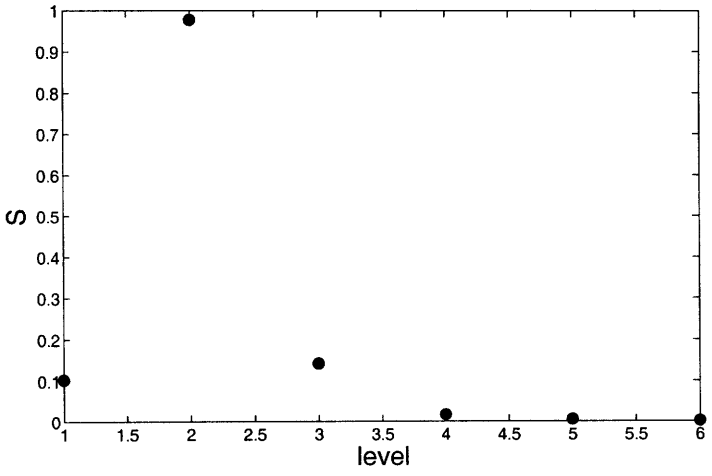

Figure 3-4: Similarity at each level of Multiscale DTSW's execution in comparing the karate punch videos

In some cases, even though Multiscale DTSW has reached the optimal level, the

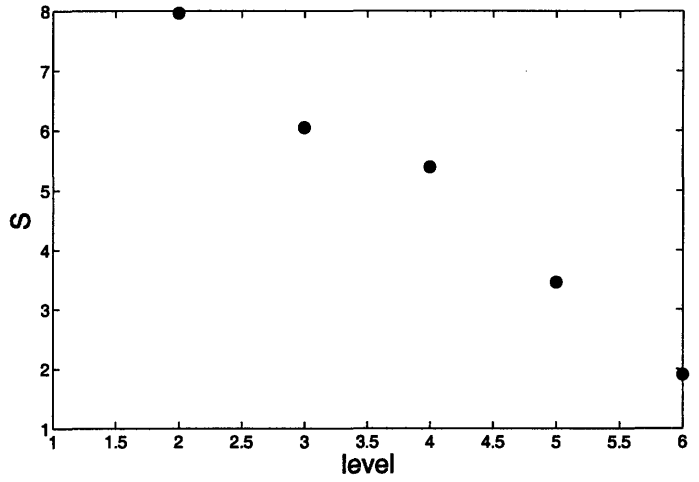

Figure 3-5: Similarity at each level of Multiscale DTSW's execution in comparing the horse racing videos

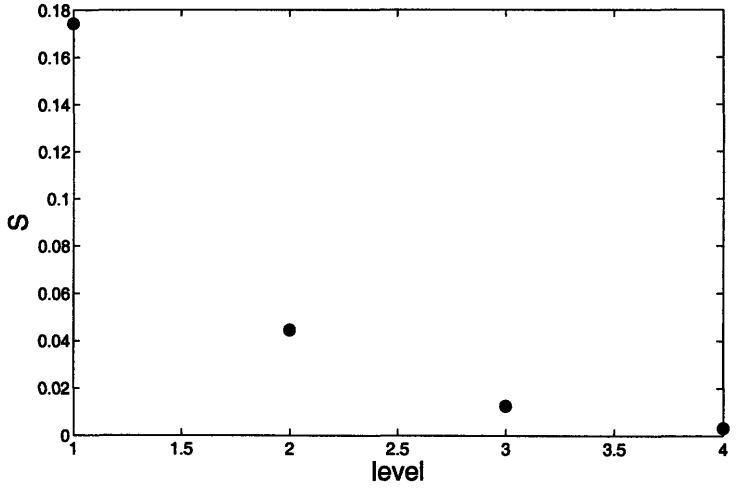

Figure 3-6: Similarity at each level of Multiscale DTSW's execution in comparing the heart valve videos

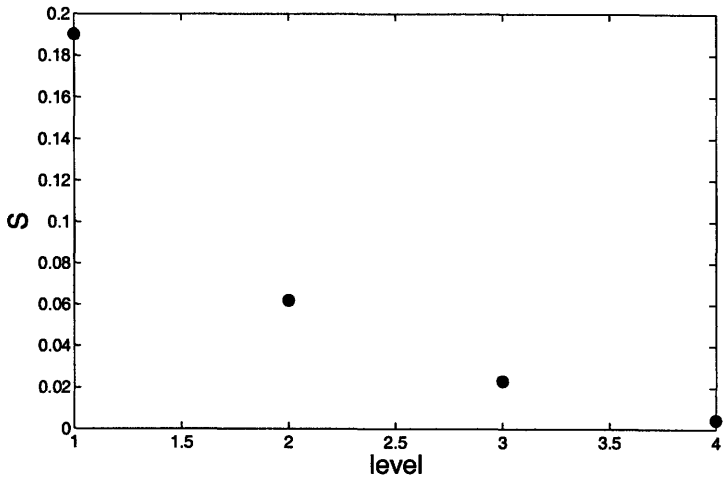

Figure 3-7: Similarity at each level of Multiscale DTSW's execution in comparing the palm opening and closing videos

value of S have not yet converged. Depend on whether we want an accurate or fast solution, we will decide if we want to continue running the Multiscale DTSW algorithm until the value of S converges or to stop when Multiscale DTSW reaches the optimal level. In this research, we chose to execute the Multiscale DTSW until it reaches the optimal level. From the experiments we conducted, by using this approach, the solution obtained by Multiscale DTSW is still within a good approximation (5% error) of the solution found by DTSW. The experimental result can be found in Section 3.4.

We use the optimal level as hard stop but check for early convergence to reduce the total number of computations, Multiscale DTSW stops and returns the value of S and the optimal warp path found so far once it detects that the value of S has converged (as shown in Figure 3-3). When the value of S has already converged, we are confident that even if we continue running the Multiscale DTSW algorithm until the optimal level, the value of S will not be substantially different and will not affect our conclusion on the similarity of the two input videos.

3.2 Complexity of Multiscale DTSW

Assume that I, J, X, and Y in the D and Cum hypervolumes are all equal to N; then the time and space complexity of Multiscale DTSW are analyzed as follows.

3.2.1 Time Complexity

- **Building hypervolumes**

  The *Elemental Distance Hypervolume (D)* and *Cumulative Distance Hypervolume (Cum)* are computed at each level, but the hypervolumes are not fully filled. The hypervolumes are partially filled based on the projected warp path from the coarser resolution optimal warp path. In addition, Multiscale DTSW also fills in any cells within r_t cells away from the projected path in the time dimension and r_s cells away from the projected path in the space dimension. The parameters r_t and r_s are the radii in the time and space dimensions that are set based on the relaxation radius specified by a user. Variables r_t and r_s determines how big the search region for finding the optimal warp path around the projected warp path is. The solution of Multiscale DTSW is more likely to be similar to the solution of DTSW when the search region is big. Therefore, the selection of the value for r_t and r_s depends on how similar a user wants the solution of Multiscale DTSW to the solution of DTSW. More explanation of r_t and r_s can be found in Section 3.3.2

  Figures 3-8 to 3-10 depicts the three basic projections of a warp path to finer resolution hypervolume. These basic projections are the same for both the time and space dimensions. The dark gray cells in the figures are the projected cells from the optimal warp path for coarser resolution. The light gray cells are the additional cells that are added to the computation of the hypervolumes based on r_t or r_s. Assume that the downsampling multiplier in time and space dimensions is two, then for a horizontal and vertical warp path, a cell in the warp path is projected into four cells at the finer resolution hypervolume. If

the downsampling multiplier in the space dimension is a and the downsampling multiplier in the time dimension is b, then each cell is projected into $a \times b$ cells. For a diagonal warp path, each cell projected is into four cells plus additional cells to smooth the connection among the projected cells on each axis.

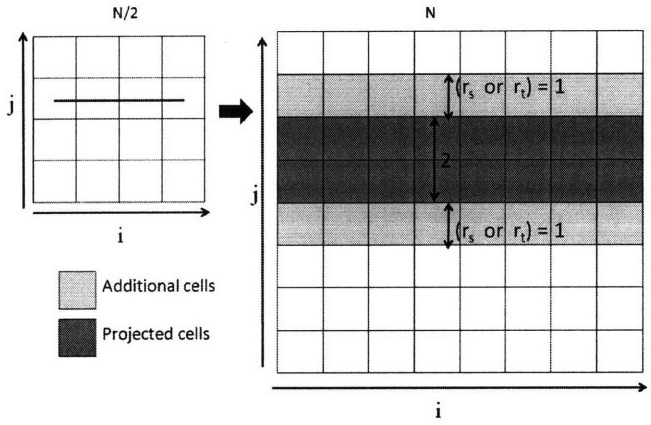

Figure 3-8: Projection of a horizontal warp path to finer resolution hypervolume with radius = 1

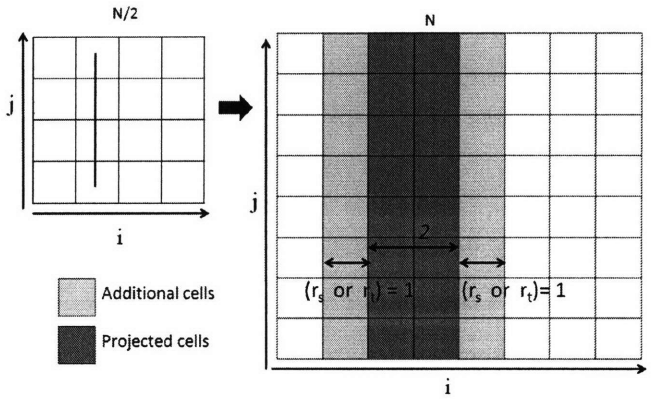

Figure 3-9: Projection of a vertical warp path to finer resolution hypervolume with radius = 1

Figure 3-11 shows an example of projecting a warp path to finer resolution hypervolume in the time dimension. From Figures 3-8 to 3-11, we observe that the number of filled cells in the finer resolution hypervolumes depends on the shape of the projected warp path. In the worst case, Multiscale DTSW will fill most of the cells of the finer resolution hypervolumes if the projected optimal warp path is a straight diagonal path in the time dimension. From Figure 3-10,

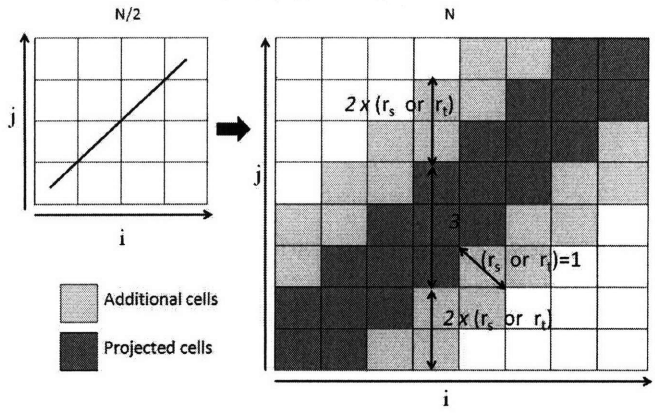

Figure 3-10: Projection of a diagonal warp path to finer resolution hypervolume with radius = 1

we can see that each column at finer resolution hypervolume (except the first and last three columns) has three projected cells and $2 \times r_t$ cells on each side of the projected cells filled.

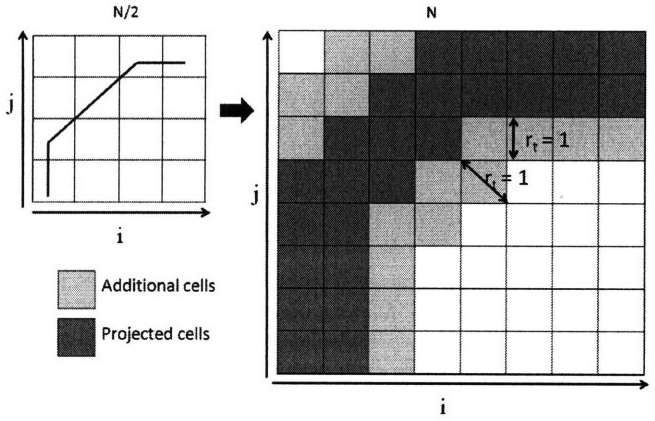

Figure 3-11: Projection of a warp path to finer resolution hypervolume in the time dimension with $r_t = 1$

Assume that all columns have three projected cells and $2 \times r_t$ cells on each side of the projected cells filled. If the length of the time dimension of the hypervolumes at a level is equal to N, then in the worst case, the number of filled cells in the time dimension of each hypervolume at that level is

$$N \times ((2 \times 2 \times r_t) + 3) = N(4r_t + 3). \tag{3.2}$$

The length of each dimension of the hypervolumes at each level is

$$\left(\frac{N}{2^m}\right)_{m=0}^{m=optimal\_level-1} = N, \frac{N}{2}, \frac{N}{2^2}, \frac{N}{2^3}, \ldots. \tag{3.3}$$

Therefore, the total cost of building each hypervolume in the time dimension for all levels is

$$\sum_{m=0}^{optimal\_level-1} \frac{N}{2^m}(4r_t+3) = N(4r_t+3)+\frac{N}{2}(4r_t+3)+\frac{N}{2^2}(4r_t+3)+\frac{N}{2^3}(4r_t+3)+\ldots. \tag{3.4}$$

Assume that the optimal level approaches infinity, then the series in Equation 3.4 is similar to the series

$$\sum_{m=0}^{\infty} \frac{1}{2^m} = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \ldots = 2. \tag{3.5}$$

Multiplying Equation 3.5 and Equation 3.2 yields

$$\begin{aligned}\sum_{m=0}^{optimal\_level-1} \frac{N}{2^m}(4r_t+3) &= N(4r_t+3) + \frac{N}{2}(4r_t+3) + \frac{N}{2^2}(4r_t+3) + \ldots \\ &= 2N(4r_t+3).\end{aligned} \tag{3.6}$$

In total, for each hypervolume, Multiscale DTSW needs $2N(4r_t+3)$ computations to build each hypervolume in the time dimension.

In the space dimension, there are two components, x and y, and Multiscale DTSW separately projects the optimal warp path from coarser resolution to finer resolution in the x and y dimensions. Two examples of projecting the optimal warp path in the x and y dimensions are shown in Figure 3-12 and Figure 3-13. Similar to the projection in the time dimension, in the worst case, the hypervolumes will be filled the most when the projected warp path is a straight diagonal path as shown in Figure 3-10. At the finer resolution, each column (except the first and last three columns) has three projected cells and $2 \times r_s$ cells on each side of the projected cells filled. Assume that the first and

last three columns also have three projected cells and $2 \times r_s$ cells on each side of the projected cells filled. Then, for each column of the hypervolumes in the x or y dimension, $4r_s + 3$ cells are filled.

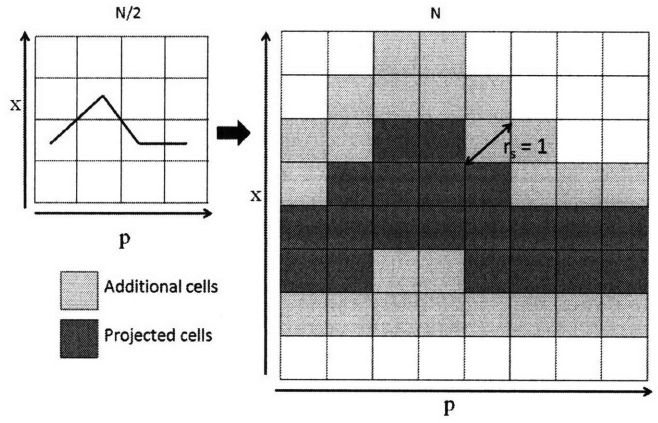

Figure 3-12: Projection of a warp path to finer resolution hypervolume in the x dimension with $r_s = 1$

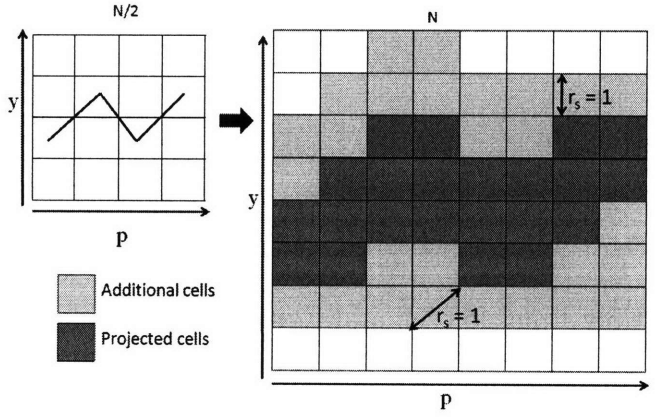

Figure 3-13: Projection of a warp path to finer resolution hypervolume in the y dimension with $r_s = 1$

Each column in the x or y dimension is the x or y axis of a cell in the time dimension. Thus, for each filled cell in the time dimension of the hypervolumes, $4r_s + 3$ cells in the x dimension and $4r_s + 3$ cells in the y dimension are also filled.

Combining the time and space dimensions, Multiscale DTSW needs

$$2N(4r_t + 3) \times (4r_s + 3)^2 \tag{3.7}$$

computations to build each hypervolume. Therefore, the total computational cost to build the *Elemental Distance* hypervolume and *Cumulative Distance* hypervolume is

$$2 \times 2N(4r_t + 3) \times (4r_s + 3)^2 = 4N(4r_t + 3)(4r_s + 3)^2. \tag{3.8}$$

- **Searching for the optimal warp path**

  Multiscale DTSW needs to search for the optimal warp path at each level. In the worst case (Equation 2.7), the optimal warp path is of length $I + J = N + N = 2N$, where N is the length of the time dimension at each level. For each element of the optimal warp path in the time dimension, Multiscale DTSW needs to compare $b_x \times b_y$ elements in the space dimension. Thus, the cost of computing the optimal warp path is

$$2N \times b_x \times b_y \tag{3.9}$$

  at each level. To sum the cost of searching for the optimal warp paths for all levels, we multiply Equation 3.9 and Equation 3.5:

$$2 \times 2N \times b_x \times b_y = 4N \times b_x \times b_y. \tag{3.10}$$

- **Projecting warp path**

  At each level, except the lowest level (level=1), Multiscale DTSW needs to project the optimal warp path from coarser resolution to finer resolution. The total computations required for projecting the optimal warp path at one level in the time dimension (assuming that the length of the time and space dimensions at that level is N) is $2N$. Likewise, the computations required for projecting

the optimal warp path at one level in the space dimension is $4N$: $2N$ in the x dimension and $2N$ in the y dimension. Therefore, the total computations required for projecting the optimal warp path at one level is

$$2N + 2N + 2N = 6N. \tag{3.11}$$

The total computations required for projecting the optimal warp paths for all levels can be computed by multiplying Equation 3.11 and Equation 3.5:

$$2 \times 6N = 12N. \tag{3.12}$$

The total time complexity of Multiscale DTSW can be computed by summing Equation 3.12, Equation 3.10, and Equation 3.8:

$$4N(4r_t + 3)(4r_s + 3)^2 + (4N \times b_x \times b_y) + 12N. \tag{3.13}$$

3.2.2 Space Complexity

Multiscale DTSW needs to store the *Elemental Distance* and *Cumulative Distance* hypervolumes. The space complexity of storing the hypervolumes is the same as the time complexity of building the hypervolumes (Equation 3.8): $4N(4r_t + 3)(4r_s + 3)^2$. Multiscale DTSW also needs to store the optimal warp path at each level. At each level, the optimal warp path consists of four elements (i, j, x, y), and the longest optimal warp path is $2N$ long. Therefore, the total space needed to store the optimal warp path at each level is

$$4 \times 2N = 8N. \tag{3.14}$$

The total space needed to store the optimal warp paths for all levels is computed by multiplying Equation 3.14 with Equation 3.5:

$$2 \times 8N = 16N. \tag{3.15}$$

Therefore, the total space complexity is

$$4N(4r_t + 3)(4r_s + 3)^2 + 16N. \tag{3.16}$$

In conclusion, Multiscale DTSW has $O(Nr_t r_s^2)$ time and space complexity. In the best case, when r_t and r_s are relatively much smaller than N, Multiscale DTSW has linear time and space complexity ($O(N)$).

3.3 Analysis of Multiscale DTSW Algorithm

In this section, we describe the time complexity of Multiscale DTSW with n levels in more details. We then explain about the optimal level and relaxation radius followed by the efficiency of Multiscale DTSW.

The details of the total time complexity of Multiscale DTSW with n levels are elaborated as follows. As explained before, our convention is that level 1 is for the coarsest resolution and level n is for the finest resolution.

- **Computational Cost at the Coarsest Resolution (Level 1)**

  At the coarsest resolution, Multiscale DTSW needs to perform the full DTSW algorithm, or Multiscale DTSW needs to compute every cells in the coarsest resolution hypervolumes. The computational cost of Multiscale DTSW at the coarsest resolution is the computational cost of building the hypervolumes and the cost of searching for the optimal warp path. The length of each dimension of the hypervolumes at level=1 is $\frac{N}{2^{n-1}}$. Therefore, the computational cost is

  $$2 \times \left(\frac{N}{2^{n-1}}\right)^4 + 2\left(\frac{N}{2^{n-1}}\right) b_x b_y. \tag{3.17}$$

  Variables b_x and b_y are the maximum allowable changes in the x and y dimensions of the optimal warp path between a frame and its adjacent frame. They are constant for each set of input videos.

- **Computational Cost at Resolution other than the Coarsest Resolu-**

tion (Level 2...n)

- Based on Equation 3.7, the total computational cost of building the *Elemental Distance* and *Cumulative Distance* hypervolumes for all the levels except level one is

$$2(4r_t+3)(4r_s+3)^2 \times N(1+\frac{1}{2}+\frac{1}{2^2}+...\frac{1}{2^{n-2}}). \tag{3.18}$$

- Based on Equation 3.9, the total computational cost of finding the optimal warp path for all the levels except level one is

$$2 \times b_x \times b_y \times N(1+\frac{1}{2}+\frac{1}{2^2}+...\frac{1}{2^{n-2}}). \tag{3.19}$$

- Based on Equation 3.11, the total computational cost of projecting the optimal warp path from coarser resolution to finer resolution for all levels except level one is

$$6N(1+\frac{1}{2}+\frac{1}{2^2}+...\frac{1}{2^{n-2}}). \tag{3.20}$$

Combining all together, the total computations of Multiscale DTSW with n levels is

$$\begin{aligned} Cost(n) &= 2\left(\frac{N}{2^{n-1}}\right)^4 + 2\left(\frac{N}{2^{n-1}}\right) b_x b_y + 2(4r_t+3)(4r_s+3)^2 N\left(\sum_{i=0}^{i=n-2}\frac{1}{2^i}\right) + \\ &\quad 2b_x b_y N\left(\sum_{i=0}^{i=n-2}\frac{1}{2^i}\right) + 6N\left(\sum_{i=0}^{i=n-2}\frac{1}{2^i}\right) \\ &= 2\left(\frac{N}{2^{n-1}}\right)^4 + 2\left(\frac{N}{2^{n-1}}\right) b_x b_y + \\ &\quad \left(\sum_{i=0}^{i=n-2}\frac{1}{2^i}\right)(2N(4r_t+3)(4r_s+3)^2 + 2b_x b_y N + 6N). \end{aligned} \tag{3.21}$$

Figure 3-14 shows the plot of the total computations versus number of levels of

Multiscale DTSW based on Equation 3.21 for $N = 100$ and $r_t = r_s = \{5, 10, .., 30\}$.

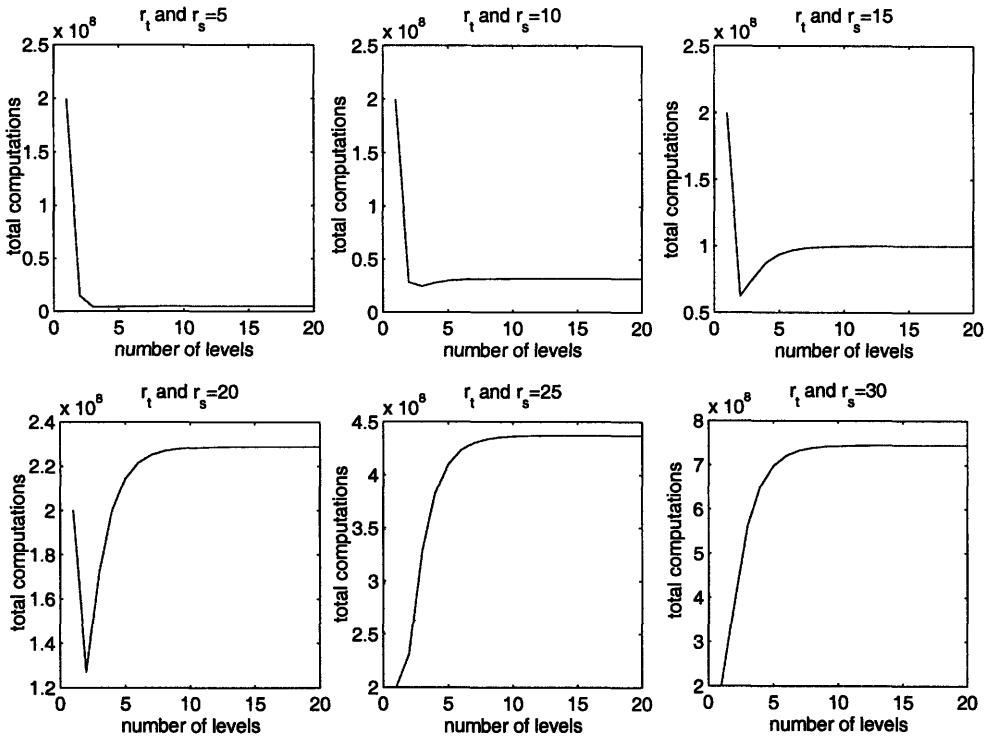

Figure 3-14: Total computations versus number of levels of Multiscale DTSW in comparing two videos with the length of each dimension of the *Elemental Distance* and *Cumulative Distance* hypervolumes (N) equal to 100 and $r_s = r_t = \{5, 10, 15, 20, 25, 30\}$

3.3.1 Optimal Level

From Figure 3-14, we can observe that the minimum total computation is not always achieved by running Multiscale DTSW with many levels of resolution. It may be more costly for Multiscale DTSW to use $n+1$ levels than to use n levels. We call the number of levels that minimizes the total computation as optimal level. We run the Multiscale DTSW with the optimal level of resolution to fully explore the efficiency of Multiscale DTSW.

The optimal level can be computed by solving the following optimization problem.

$$\begin{aligned} obj &= min \quad Cost(n) \\ subject \quad to \quad & n \geq 0 \\ n \quad \epsilon \quad & Z^{+}. \end{aligned} \tag{3.22}$$

The index n where $Cost(n)$ is equal to the value of obj is the optimal level.

The optimal level can also be computed by using greedy approach. In greedy approach, we first compute $C(1)$, then we compare its computation with $C(2)$. If $C(2)$ is higher than $C(1)$, we stop searching and the optimal level is one. However, if $C(2)$ is smaller than $C(1)$, we continue searching by comparing $C(2)$ with $C(3)$, $C(3)$ with $C(4)$, $C(4)$ with $C(5)$, and so on until we find that $C(n+1)$ is greater than $C(n)$ and the optimal level is n.

There are some applications that always compare two videos with a fixed length of frames or a little variation in the length of the videos. For example, the manufacturing monitoring system. The target video is a video of machinery performing one cycle of manufacturing process, and because the target video is only one, the length of the target video is constant. The query video is a real time video of machinery performing one cycle of manufacturing process. Because the amount of time to perform a cycle of manufacturing process is almost fixed, there is no much variation in the length of the query video. Therefore, in these type of applications, it is useful to predefine the optimal level to avoid repeating the computation of optimal level over and over again.

3.3.2 Relaxation Radius

Because Multiscale DTSW does not search for all possible warp paths to find the optimal warp path, we cannot guarantee that the optimal warp path found by Multiscale DTSW is the global optimal. To get a better accuracy, Multiscale DTSW needs to search for more warp paths in addition to the warp path projected from the coarser resolution optimal warp path. We have implemented this approach by defining relaxation radius (rr). Relaxation radius determines the value of r_t and r_s.

Variables r_t and r_s determines the number of additional cells that Multiscale DTSW needs to compute beyond those cells projected from the coarser resolution optimal warp path. The bigger the value of r_t or r_s is, the more warp paths that Multiscale DTSW considers, and the more confidence we have in the accuracy of the solution of Multiscale DTSW.

We consider the solution of DTSW as the global optimal solution because DTSW find the optimal warp path that has the minimum cost among all warp paths. The more similar the solution of Multiscale DTSW to the solution of DTSW, the more confidence we have in the accuracy of the solution of Multiscale DTSW.

In practice, we may not need to compute the most optimal warp path. We may only want to get a rough idea if two videos are alike. Hence, we can set the r_t and r_s to be small, and Multiscale DTSW gives the answer in a much shorter time than DTSW. If r_t and r_s are relatively much smaller than N, then Multiscale DTSW will have $O(N)$ time and space complexity while DTSW has $O(N^4)$ time and space complexity.

For a more accurate solution, we set large values for r_t and r_s. The bigger the r_t and r_s are, the more computation Multiscale DTSW requires. Multiscale DTSW may become more computationally expensive than DTSW when r_t and r_s are very large.

We want to analyze for which values of r_t and r_s that Multiscale DTSW requires more computations than DTSW. We call the value of such radius as r_t^* for the radius threshold in the time dimension and r_s^* for the radius threshold in the space dimension. If we want r_t's value to be greater than r_t^* and r_s's value to be greater than r_s^*, then we should use DTSW instead of Multiscale DTSW. On the other hand, when r_t is smaller than r_t^* or r_s is smaller than r_s^*, then we should use Multiscale DTSW instead of DTSW.

From Figure 3-14, we can observe that at $r_t = r_s = 25$ and $r_t = r_s = 30$, using one level in Multiscale DTSW (or DTSW) has less computations than using more than one level in Multiscale DTSW. We can also observe that for certain values of r_t and r_s, if the total computation of Multiscale DTSW with two levels is more than of

Multiscale DTSW with one level (DTSW), then the total computation of Multiscale DTSW with three or more levels will also be greater than the total computation of Multiscale DTSW with one level. In other words, if the total computation of Multiscale DTSW with two levels is greater than of Multiscale DTSW with one level (DTSW), we can assure that the total computation of Multiscale DTSW with any levels other than one will be greater than of Multiscale DTSW with one level (DTSW), and therefore, we should use DTSW for such r_t and r_s.

Hence, we can determine whether we should use DTSW or Multiscale DTSW based on the comparison of the complexity of Multiscale DTSW with two levels and of Multiscale DTSW with one level. We can find r_t^* and r_s^* by computing the smallest r_t and r_s such that the computation of Multiscale DTSW with two levels is greater than of Multiscale DTSW with one level.

Let $\Delta_1 = Cost(2) - Cost(1)$. Hence,

$$\begin{aligned}
\Delta_1 &= 2\left(\frac{N}{2}\right)^4 + 2\left(\frac{N}{2}\right) b_x b_y + 2N(4r_t+3)(4r_s+3)^2 + 2Nb_x b_y + 6N - (2N^4 + 2Nb_x b_y) \\
&= 2\left(\frac{1}{2^4} - 1\right) N^4 + 2(\frac{1}{2} - 1)(Nb_x b_y) + 2N(4r_t+3)(4r_s+3)^2 + 2Nb_x b_y + 6N \\
&= 2\left(\frac{-15}{16}\right) N^4 - Nb_x b_y + 2N(4r_t+3)(4r_s+3)^2 + 2Nb_x b_y + 6N \\
&= \left(\frac{-15}{8}\right) N^4 - Nb_x b_y + 2N(4r_t+3)(4r_s+3)^2 + 2Nb_x b_y + 6N
\end{aligned} \tag{3.23}$$

We are looking for smallest r_t and r_s such that $\Delta_1 \geq 0$. In other words, we are looking for smallest r_t and r_s that satisfy

$$\begin{aligned}
\Delta_1 &\geq 0 \\
2N(4r_t+3)(4r_s+3)^2 &\geq \frac{15}{8}N^4 + Nb_x b_y - 2N(b_x b_y + 3) \\
(4r_t+3)(4r_s+3)^2 &\geq \frac{1}{2}\left(\frac{15}{8}N^3 + b_x b_y - 2(b_x b_y + 3)\right).
\end{aligned} \tag{3.24}$$

Variables r_t^* and r_s^* are dependent variables. If we let r_t to be a constant, then we can compute r_s^* analytically from Equation 3.24 or r_s^* is the smallest r_s that satisfies

$$\begin{aligned}(4r_s+3) &\geq \sqrt{\frac{\frac{1}{2}(\frac{15}{8}N^3+b_xb_y-2(b_xb_y+3))}{4r_t+3}} \\ r_s &\geq \frac{\sqrt{\frac{\frac{1}{2}(\frac{15}{8}N^3+b_xb_y-2(b_xb_y+3))}{4r_t+3}}-3}{4}.\end{aligned} \tag{3.25}$$

If we let r_s to be a constant, then we can compute r_t^* analytically from Equation 3.24 or r_t^* is the smallest r_t that satisfies

$$\begin{aligned}(4r_t+3) &\geq \frac{\frac{1}{2}(\frac{15}{8}N^3+b_xb_y-2(b_xb_y+3))}{(4r_s+3)^2} \\ r_t &\geq \frac{\frac{\frac{1}{2}(\frac{15}{8}N^3+b_xb_y-2(b_xb_y+3))}{(4r_s+3)^2}-3}{4}.\end{aligned} \tag{3.26}$$

Otherwise, if we let r_t and r_s to be equal, then the value of r_t^* and r_s^* is equal to the value of r that satisfies

$$\begin{aligned}(4r+3) &\geq \sqrt[3]{\frac{1}{2}\left(\frac{15}{8}N^3+b_xb_y-2(b_xb_y+3)\right)} \\ r &\geq \frac{\sqrt[3]{\frac{1}{2}\left(\frac{15}{8}N^3+b_xb_y-2(b_xb_y+3)\right)}-3}{4}.\end{aligned} \tag{3.27}$$

We can also compute r_t^* and r_s^* from Equation 3.24 if we let $r_t = 2 \times r_s$ or any other equation governing the relationship between r_t and r_s.

In summary, if we know the relationship between r_t and r_s or if we fix r_t or r_s, we can compute r_t^* and r_s^* from Equation 3.24.

Let rr be a variable for the probability of the solution of Multiscale DTSW being similar to the solution of DTSW. The relaxation radius of 1 means that Multiscale DTSW guarantees that the solution of Multiscale DTSW is exactly the same as the solution of DTSW. The relaxation radius of 0.8 means that the solution of Multiscale DTSW has 80% probability of being similar to the solution of DTSW.

Based on the relaxation radius that a user wants, r_s for the Multiscale DTSW

algorithm will be

$$r_s = rr \times r_s^*. \tag{3.28}$$

Likewise, r_t will be

$$r_t = rr \times r_t^*. \tag{3.29}$$

If a user wants the global optimal solution, r_t will be equal to r_t^* and r_s will be equal to r_s^*. And since the both radii pass their corresponding thresholds (r_t^* and r_s^*), Multiscale DTSW uses the full DTSW algorithm and hence, the user obtains the global optimal solution.

In the case where a user inputs certain values for r_t and r_s instead of a relaxation radius, Multiscale DTS substitutes the value of r_t and r_s into Equation 3.24 and decides if Multiscale DTSW will use DTSW (Multiscale DTSW with one level) or Multiscale DTSW with more than one level.

Figure 3-15 shows the optimal level of Multiscale DTSW in comparing videos with $N = 100$ versus relaxation radius $(rr) = \{0...1\}$. The graph looks like a step function. When the relaxation radius is small, the optimal level is big. It means that Multiscale DTSW can use many levels of resolution while still reducing the total amount of computations required. However, when the relaxation radius is big, the radius will be big, and the hypervolumes at the finest resolution is almost full. Multiscale DTSW can only downsample the input videos to coarser resolution if the total computations at the coarser resolution is still less than the amount of computations needed for the empty cells of the finest resolution hypervolumes. Since the radius (r_t or r_s) is big, the hypervolumes at the coarser resolution is also almost full and the computational cost is expensive. Therefore, it is not efficient to use many levels in Multiscale DTSW when the relaxation radius is big.

3.3.3 Multiscale DTSW Efficiency

The time complexity of Multiscale DTSW is computed based on the total number of computations required and we normalize it by dividing it with the time complexity of DTSW. The normalized time complexity ranges between 0 and 1. The normalized

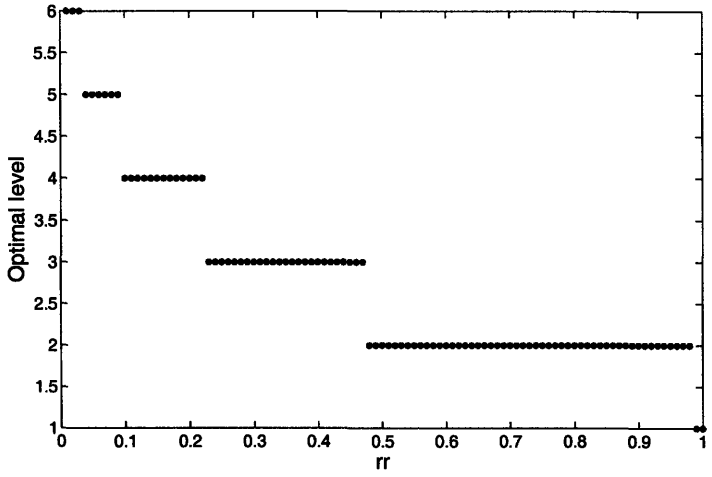

Figure 3-15: Optimal level versus relaxation radius of Multiscale DTSW in comparing two videos with N=100

time complexity cannot go beyond one since when Multiscale DTSW is more computationally expensive than DTSW, we will use DTSW. Therefore, the maximum value for the normalized time complexity is one.

Figure 3-16 shows the plot of the normalized time complexity of Multiscale DTSW in comparing two videos with various lengths and relaxation radius ranges between 0 and 1. We can see that the normalized time complexity plot lies below the diagonal axis of the plot. This means that to obtain an x relaxation radius, Multiscale DTSW needs less than x percentage of computational effort of DTSW.

Figure 3-17 shows the normalized time complexity of Multiscale DTSW in comparing videos with various lengths for relaxation radius = 0.5, 0.6, 0.7, and 0.8. The figure shows that for a constant relaxation radius, the normalized time complexity of Multiscale DTSW is getting smaller and smaller as N is getting bigger and bigger.

The time complexity of Multiscale DTSW is $O(Nr_t r_s^2)$. If we fix r_t and r_s, we can assume that the time complexity of Multiscale DTSW is $O(N)$. We normalized it with the time complexity of DTSW, which is $O(N^4)$. Therefore, the normalized time complexity of Multiscale DTSW is $(\frac{O(N)}{O(N^4)}) = O(\frac{1}{N^3})$. Therefore, as N is getting bigger, the normalized time complexity of Multiscale DTSW is getting smaller. This

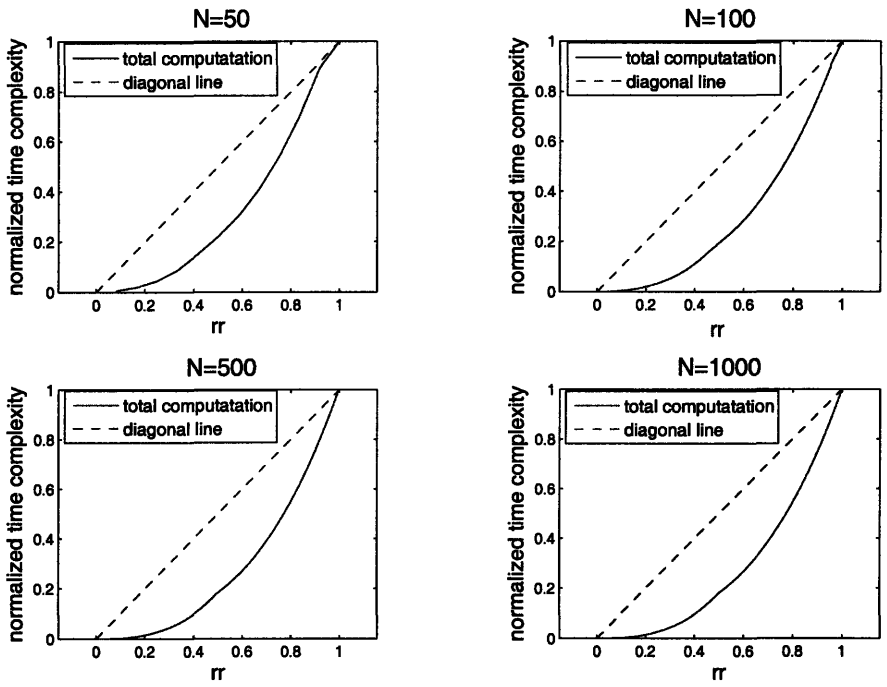

Figure 3-16: Normalized time complexity of Multiscale DTSW versus relaxation radius in comparing two videos with N={50, 100, 500, 1000}

observation is useful since we want to apply Multiscale DTSW to large videos (large N).

3.4 Experimental Result

Table 3.1 shows the experimental results of running Multiscale DTSW and DTSW algorithms implemented in Matlab on various target videos (Appendix A) using an IBM Pentium M laptop (1.3 GHz processor with 768 MB memory). The table shows that for small relaxation radius, the execution time of Multiscale DTSW is much faster than of DTSW. For big relaxation radius, the execution time of Multiscale DTSW is almost equal or equal to of DTSW. These results comply with our analytical analysis of the complexity of Multiscale DTSW and DTSW, which is $O(Nr_t r_s^2)$ and $O(N^4)$, respectively. For small relaxation radius, the value of r_s and r_t are relatively small as compared to the value of N, so the complexity of Multiscale DTSW, which is

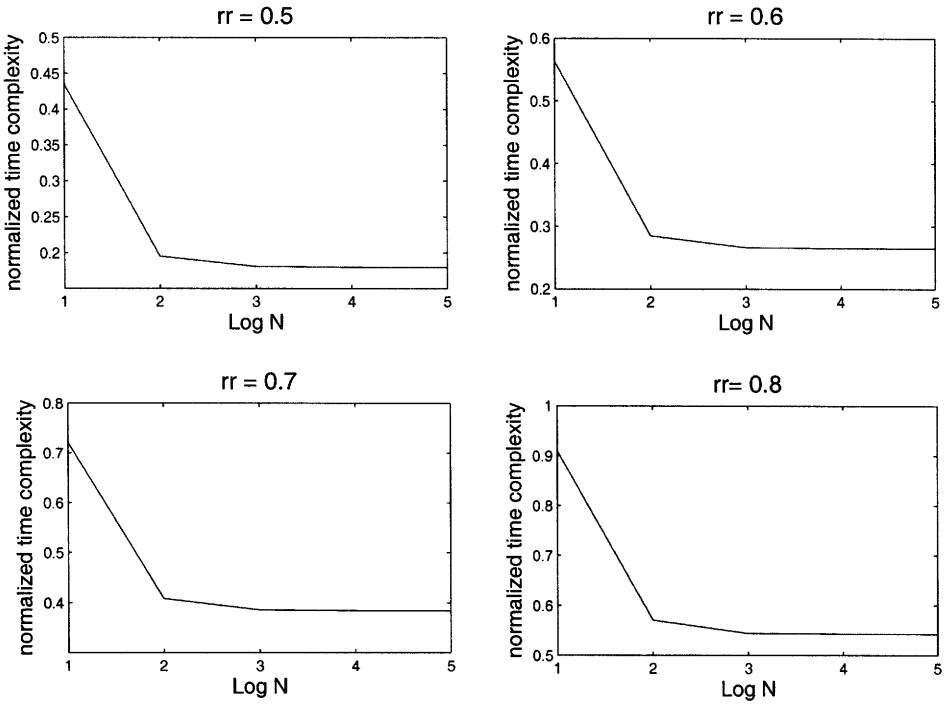

Figure 3-17: Normalized time complexity of Multiscale DTSW versus log N with relaxation radius = {0.5, 0.6, 0.7, 0.8}

$O(Nr_t r_s^2)$, is smaller than of DTSW, which is $O(N^4)$. Therefore, the execution time of Multiscale DTSW is much faster. For large relaxation radius, $O(Nr_t r_s^2)$ could be as big as $O(N^4)$, and therefore, the execution time of Multiscale DTSW is almost equal to of DTSW.

Table 3.2 shows the normalized execution time for the corresponding execution time in Table 3.1. To obtain a normalized execution time, we divided the execution time with the corresponding execution time of DTSW.

Table 3.3 shows the comparison of error of Multiscale DTSW in comparing the video sets in Table 3.1. The error is computed as

$$\%Error = \frac{|S_{MultiscaleDTSW} - S_{DTSW}|}{S_{DTSW}} \times 100\%. \tag{3.30}$$

For a relaxation radius of at least 60%, the error of Multiscale DTSW is less than 5%, except for the random video. The large error in the results of the random

Table 3.1: Comparison of execution time of DTSW and Multiscale DTSW for various relaxation radii on various sets of video, running on an IBM Pentium M laptop (1.3 GHz processor with 768 MB memory)

Video set	Dimension	Execution time of DTSW in seconds	Execution time of Multiscale DTSW in seconds			
			rr = 0.2	rr = 0.4	rr = 0.6	rr = 0.8
Heart valve	40 × 46 × 8 × 11	344.02	109.35	115.57	167.2	316.37
	40 × 46 × 16 × 21	1912.15	291.6	488.7	698.44	920.82
Karate punch	41 × 45 × 12 × 7	288.6	56.61	69.17	138.7	268.022
	81 × 89 × 26 × 13	7619.94	641.39	1041.34	1589.77	3431.14
Random	37 × 25 × 7 × 7	73.64	12.93	16.69	39.87	71.15
Horse racing	30 × 27 × 37 × 13	88.13	12.91	14.18	27.91	57.33
	60 × 53 × 73 × 26	1473.96	83.79	108.2	228.45	606.7
Palm opening and closing	43 × 38 × 14 × 6	132.73	37.03	47.43	70.5	129.94
	85 × 75 × 30 × 10	3020.36	227.29	341.26	489.46	1538.56
Person Walking	24 × 13 × 70 × 36	1264.45	12.12	18.34	36.52	167.3

videos' similarity could be due to the randomness of the synthetic random video that we developed. The random video (Appendix A) was designed by using a random function in Matlab with several constraints that limit its variation from frame to frame.

For the karate punch video set, we could achieve a 0% error for the Multiscale DTSW solution with the execution time 0.21 of the execution time of DTSW.

Table 3.4 to Table 3.6 show the experimental results of running Multiscale DTSW and DTSW implemented in Matlab on various target videos (Appendix A) using a Gateway PC (3.4 GHz with 1 GB memory). The experimental results further support the efficiency of Multiscale DTSW. With Multiscale DTSW, we can obtain a comparison result with less than 0.78% error with an execution time of less than 6% of the the execution time of DTSW. By observing Table 3.5 and Table 3.2, we can observe that the amount of computation saved by Multiscale DTSW for large data (large dimensions) is more than the amount of computation saved by Multiscale DTSW for small data (small dimensions). These experimental results support our claim in Section 3.3.3 that the normalized time complexity of Multiscale DTSW is getting smaller for bigger dimension videos for a constant relaxation radius. For the

Table 3.2: Comparison of normalized execution time of Multiscale DTSW for various relaxation radii on various sets of video, running on an IBM Pentium M laptop (1.3 GHz processor with 768 MB memory)

Video set	Dimension	Normalized execution time of Multiscale DTSW			
		rr = 0.2	rr = 0.4	rr = 0.6	rr = 0.8
Heart valve	40 × 46 × 8 × 11	0.32	0.34	0.49	0.92
	40 × 46 × 16 × 21	0.15	0.26	0.37	0.48
Karate punch	41 × 45 × 12 × 7	0.2	0.24	0.48	0.93
	81 × 89 × 26 × 13	0.08	0.14	0.21	0.45
Random	37 × 25 × 7 × 7	0.18	0.23	0.54	0.97
Horse racing	30 × 27 × 37 × 13	0.15	0.16	0.32	0.65
	60 × 53 × 73 × 26	0.06	0.07	0.16	0.41
Palm opening and closing	43 × 38 × 14 × 6	0.28	0.36	0.53	0.98
	85 × 75 × 30 × 10	0.08	0.11	0.16	0.51
Person walking	24 × 13 × 70 × 36	0.01	0.01	0.03	0.13

karate punch video, for the dimension of $41 \times 45 \times 12 \times 7$ and relaxation radius = 0.2, the normalized time complexity is 0.2. However, for the same video set and relaxation radius but with the dimension of $81 \times 89 \times 26 \times 13$, the normalized time complexity is 0.08.

Figure 3-18 and Figure 3-19 show the execution time of Multiscale DTSW running on a Gateway PC (3.4 GHz processor with 1 GB memory) for comparing the karate punch videos with dimension = $81 \times 89 \times 26 \times 13$ and $r_t = r_s = \{5, 10, ..., 35\}$. We can observe that the optimal level for Multiscale DTSW varies depending on the configuration of r_t and r_s. It is not true that the more the number of levels that Multiscale DTSW uses, the less computations that Multiscale DTSW requires. The experimental results shown in these figures support our analytical result explained in Section 3.3.1 and depicted in Figure 3-14.

For some videos, if we know that the low resolution videos are a good representation of the high resolution videos (no aliasing), we can fix the r_s and r_t for all levels of Multiscale DTSW in comparing the videos. In our experiment, the karate punch videos do not suffer aliasing when we downsample them. Therefore, we can fix r_t and r_s for all levels of Multiscale DTSW. Figure 3-20 shows the execution time of Multiscale DTSW and DTSW for various resolutions of karate punch videos with

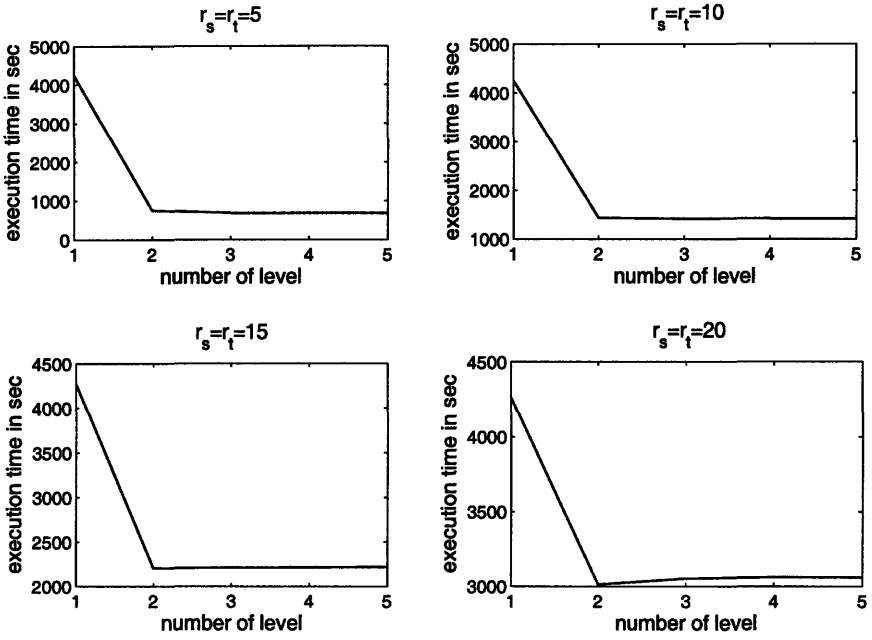

Figure 3-18: Execution time of Multiscale DTSW running on a PC with 3.4 GHz processor and 1 GB memory versus number of levels in comparing the karate punch videos with dimension $= 81 \times 89 \times 26 \times 13$ and $r_t = r_s = \{5, 10, 15, 20\}$

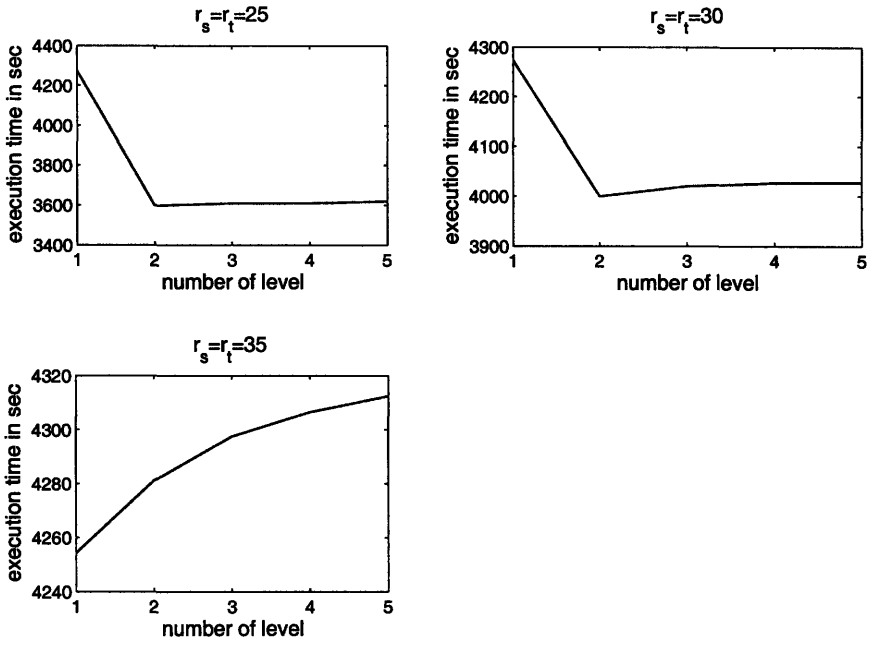

Figure 3-19: Execution time of Multiscale DTSW running on a PC with 3.4 GHz processor and 1 GB memory versus number of levels in comparing the karate punch videos with dimension = $81 \times 89 \times 26 \times 13$ and $r_t = r_s = \{25, 30, 35\}$

Table 3.3: Comparison of error of Multiscale DTSW for various relaxation radii on various sets of video, running on an IBM Pentium M laptop (1.3 GHz processor with 768 MB memory)

Video set	Dimension	Error of Multiscale DTSW in percentage			
		rr = 0.2	rr = 0.4	rr = 0.6	rr = 0.8
Heart valve	40 x 46 x 8 x 11	15.8	15.8	3.67	0.02
	40 x 46 x 16 x 21	5.28e10	5.28e10	0	0
Karate punch	41 x 45 x 12 x 7	11	8.95	0	0
	81 x 89 x 26 x 13	4.5e5	0.01	0	0
Random	37 x 25 x 7 x 7	1e5	1e5	3.9e4	9.3e3
Horse racing	30 x 27 x 37 x 13	36.65	13.66	3.5	1.92
	60 x 53 x 73 x 26	5.81	5.48	4.5	0
Palm opening and closing	43 x 38 x 14 x 6	2.46	0	0	0
	85 x 75 x 30 x 10	2.28e10	0.12	0.36	0
Person walking	24 x 13 x 70 x 36	10.75	10.48	1.47	0

Table 3.4: Comparison of execution time of DTSW and Multiscale DTSW for various relaxation radii in comparing the karate punch and horse racing videos running on a PC with 3.4 GHz processor and 1 GB memory

Video set	Dimension	Execution time of DTSW in seconds	Execution time of Multiscale DTSW in seconds			
			rr = 0.2	rr = 0.4	rr = 0.6	rr = 0.8
Karate punch	81 x 89 x 52 x 28	79771.64	4461.04	7209.61	13745.22	21044.25
	162 x 177 x 26 x 13	24547.13	1566.77	2501.44	4900.59	7734.03
Horse racing	119 x 105 x 145 x 50	12034.59	331.59	952.03	2075.14	3260.30

$r_s = r_t = 5$. From the figure, we can observe that the time complexity of DTSW is exponential while the time complexity of Multiscale DTSW is linear. This observation supports our analysis in Section 3.3.3 that for constant r_t and r_s, the time complexity of Multiscale DTSW is $O(N)$, while the time complexity of DTSW is $O(N^4)$.

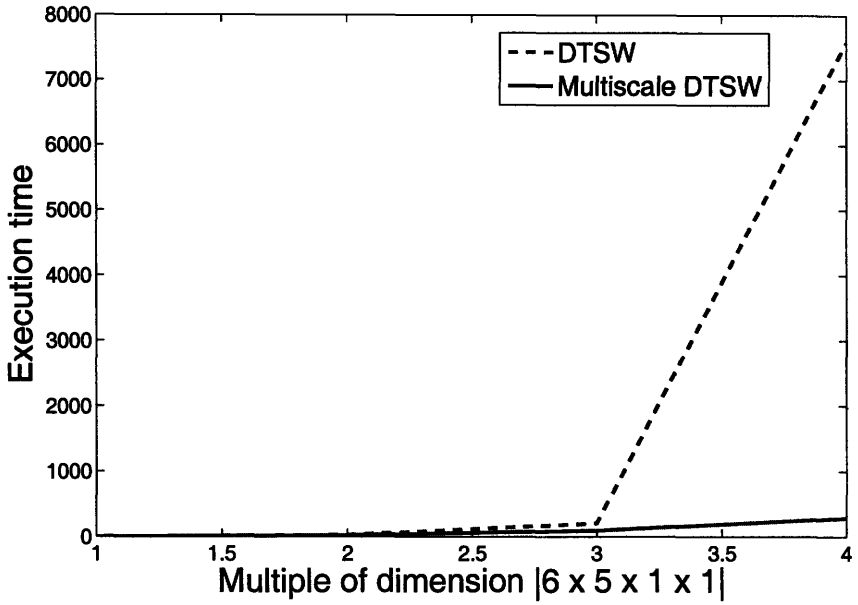

Figure 3-20: Execution time of DTSW and Multiscale DTSW running on a PC with 3.4 GHz processor and 1GB memory in comparing the karate punch videos with $r_s = r_t = 5$

Table 3.5: Comparison of normalized execution time of DTSW and Multiscale DTSW for various relaxation radii in comparing the karate punch and horse racing videos running on a PC with 3.4 GHz processor and 1 GB memory

Video set	Dimension	Normalized execution time of Multiscale DTSW			
		rr = 0.2	rr = 0.4	rr = 0.6	rr = 0.8
Karate punch	81 × 89 × 52 × 28	0.056	0.09	0.17	0.26
	162 × 177 × 26 × 13	0.06	0.1	0.2	0.32
Horse racing	119 × 105 × 145 × 50	0.03	0.08	0.17	0.27

Table 3.6: Comparison of error of DTSW and Multiscale DTSW for various relaxation radii in comparing the karate punch and horse racing videos running on a PC with 3.4 GHz processor and 1 GB memory

Video set	Dimension	Error of Multiscale DTSW in percentage			
		rr = 0.2	rr = 0.4	rr = 0.6	rr = 0.8
Karate punch	81 × 89 × 52 × 28	0.778	0	0	0
	162 × 177 × 26 × 13	0	0	0	0
Horse racing	119 × 105 × 145 × 50	0.77	0.04	0	0

Chapter 4

Extension of Multiscale DTSW

The computational cost of Multiscale DTSW can be further reduced by using the following techniques.

4.1 Multiscale DTSW with Eigenframes Implementation

4.1.1 Principal Component Analysis

Most of the query videos compared by Multiscale DTSW are videos of some sequence of action. We expect small frame to frame changes in the query video. This suggests that a small number of frames can be used to approximate the original query frames. We use Principal Component Analysis (PCA), whose details can be found in the literature [25, 26].

We focus on how the principal component analysis is used to reduce the computation in Multiscale DTSW. The use of PCA in DTSW is discussed in Anthony [1]. The details of computing the principal component can be found in [25, 26].

The principal components are the orthonormal basis set of the covariance of the original data. The principal components are also the eigenvector of the covariance matrix of the data. In the literature, the principal components for image analysis are called the Eigenframes. The frames in the Eigenframes are ordered from the most

dominant frame to the least dominant frame. The most dominant frame represents the majority of the variation in the query video, and the second most dominant frame represents the second largest amount of variation in the query video.

The number of frames in the Eigenframes set is the same as the number of frames in the query video. Assume that the query video is of length J frames, and the set of frames of the query video = $\{Q_1, Q_2, Q_3, \ldots, Q_J\}$. The set of Eigenframes = $\{E_1, E_2, E_3, \ldots, E_J\}$. The sequence of original query frames can be reconstructed by linearly combining all frames in the Eigenframes:

$$\begin{aligned} Q_1 &= e_{11}E_1 + e_{12}E_2 + e_{13}E_3 + \ldots + e_{1J}E_J \\ Q_2 &= e_{21}E_1 + e_{22}E_2 + e_{23}E_3 + \ldots + e_{2J}E_J \\ &\vdots \\ Q_J &= e_{J1}E_1 + e_{J2}E_2 + e_{J3}E_3 + \ldots + e_{JJ}E_J \end{aligned}$$

where e_{mn} is the projection coefficient of query frame m to Eigenframe n.

If we do not want to fully recover the original set of frames, but only an approximation to the original set, for example we only need to capture 90% of the variance in the query video, the amount of frames in the Eigenframes used to reconstruct the query frames can be reduced. For example, to capture 90% variation in the query video, we only need P most dominant Eigenframes, $P \leq J$. The query videos can be approximately represented as

$$\begin{aligned} Q_1 &\approx e_{11}E_1 + e_{12}E_2 + e_{13}E_3 + \ldots + e_{1P}E_P \\ Q_2 &\approx e_{21}E_1 + e_{22}E_2 + e_{23}E_3 + \ldots + e_{2P}E_P \\ &\vdots \\ Q_J &\approx e_{J1}E_1 + e_{J2}E_2 + e_{J3}E_3 + \ldots + e_{JP}E_P \end{aligned}$$

The less variance we chose to retain, the fewer the number of Eigenframes required.

4.1.2 Implementation of Eigenframes in Multiscale DTSW

Figure 4-1 depicts the DTSW algorithm in computing the *Elemental Distance* hypervolume. For two videos with a length of I and J frames, respectively, DTSW needs to perform $I \times J$ normalized correlation operations to compute the hypervolume.

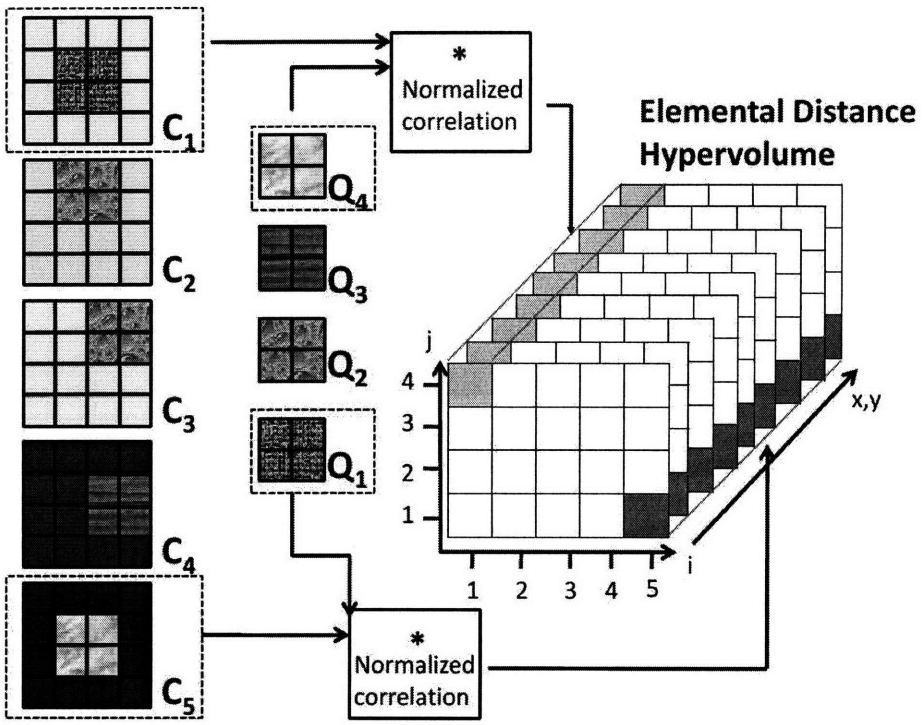

Figure 4-1: Computing the *Elemental Distance* hypervolume in the DTSW algorithm

Figure 4-2 depicts the DTSW algorithm with the Eigenframes implementation. The steps are as follows.

1. Decide the amount of variation in the query video to be captured in the Eigenframes representation.

2. Compute the Eigenframes of the query video as well as the projection coefficients of the Eigenframes that are used to recover the query frames. If we need P Eigenframes to capture the stated variance, then there are P projection coefficients for each query frame.

3. Find the normalized correlation between each frame in the target video and each of the P Eigenframes. This is $(P \times I)$ normalized correlation operations.

4. Compute the *Elemental Distance* hypervolume. Compute the hypervolume's cells between frame Q_j of the query video and frame C_i of the target video. The algorithm linearly combines the normalized correlated frames between C_i and the Eigenframes using the projection coefficient of the Eigenframes for

Q_j. The amount of computation required is $(P \times X \times Y)$ multiplications and $((P-1) \times X \times Y)$ additions. Variables X and Y are the x and y dimensions of each of the normalized correlated frames between the target video frames and the Eigenframes.

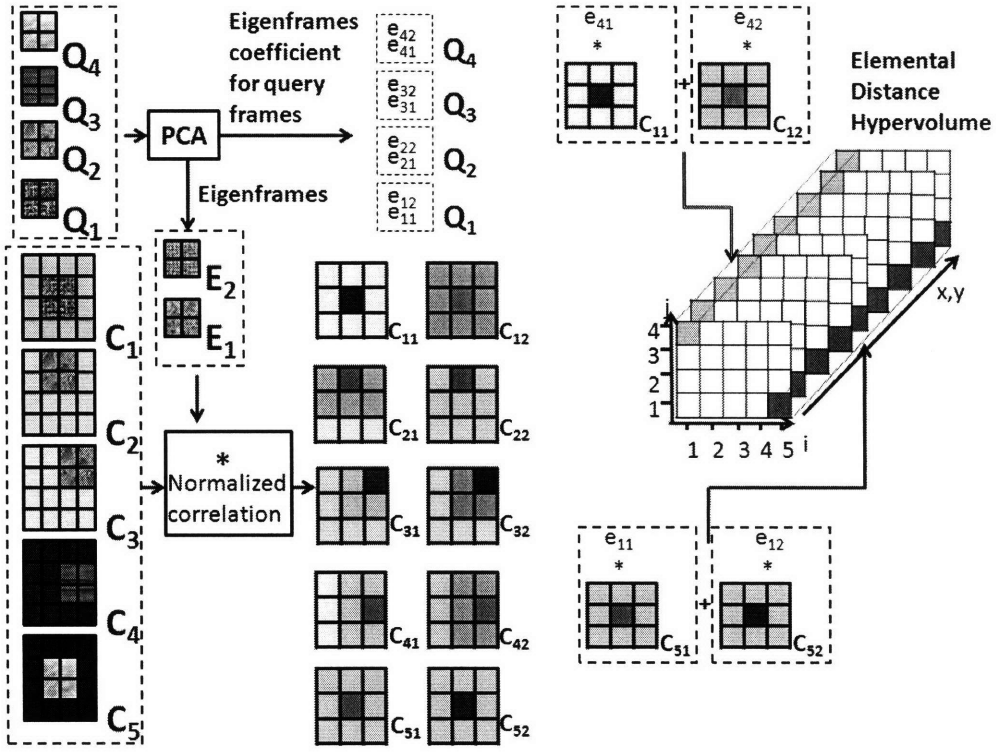

Figure 4-2: Computing the *Elemental Distance* hypervolume in the DTSW algorithm with Eigenframes implementation

Assume that the total computation to compute the Eigenframes is A computations, then the total computation of the DTSW algorithm with Eigenframes implementation is

$$A + (P \times I) \text{ normalized correlation operations} + (P \times X \times Y) \times (I \times J) \text{ multiplications} + ((P-1) \times X \times Y) \times (I \times J) \text{ additions.}$$

If we use a 100% variance, then P will be equal to J and the total computation becomes $A + (J \times I)$ normalized correlation operations$+(J \times X \times Y) \times (I \times J)$multiplications $+((J-1) \times X \times Y) \times (I \times J)$ additions. Hence, if we use DTSW algorithm with Eigenframes implementation, then the total computation will be

more than of DTSW algorithm. But if $A + (J \times X \times Y) \times (I \times J)$ multiplication $+((J-1) \times X \times Y) \times (I \times J)$ addition operations are relatively much smaller than the computation of $(J \times I)$ normalized correlation operations, then the additional computations are negligible.

Typically we use less than 100% variance, then P will be less than J. In general, $A + (P \times X \times Y) \times (I \times J)$ multiplication $+((P-1) \times X \times Y) \times (I \times J)$ addition operations are less than $(J-P) \times I$ normalized correlation operations, then by using Eigenframes implementation, we will save some computations.

4.1.3 Experimental Result

As discussed in the previous subsection, the execution time of DTSW using Eigenframes implementation in general is faster than of not using the Eigenframes method. For simplicity, we will call the DTSW without Eigenframes implementation as DTSW and DTSW with Eigenframes implementation as DTSWEF. Table 4.1 shows the normalized execution time of DTSW and of DTSWEF in comparing the karate punch, heart valve, and palm opening and closing videos. The normalized execution time is computed by dividing the execution time with the corresponding execution time of DTSW. Hence, the normalized execution time of DTSW is 1. The normalized execution time that is less than 1 means that the execution time is faster than the execution time of DTSW. Similarly, the normalized execution time that is more than 1 means that the execution time is slower than the execution time of DTSW. The experiment results support the conclusion that the computation of DTSW can be reduced by using Eigenframes implementation with a variance of less than 100%. In comparing all of the videos, the normalized execution time of DTSWEF with a variance of less than 100% is less than 1. With a 100% variance, the execution time of DTSWEF may be slower than of DTSW, as shown in the execution time of DTSWEF in comparing the palm opening and closing video.

Table 4.2 shows the error of the DTSWEF results as compared to the results of DTSW. The error is calculated by subtracting the 3-D Euclidean distance between the warped query video and the warped target video found using DTSW from the

Table 4.1: The normalized execution time of DTSWEF in comparing the karate punch, heart valve, and palm opening and closing videos with the variance = {100%, 90%, 80%, 70%}

Video set	Dimension	Normalized execution time of DTSW	Normalized execution time of DTSWEF			
			var=100	var=90	var=80	var=70
Karate punch	41 × 45 × 12 × 7	1	0.99	0.81	0.78	0.74
Heart valve	40 × 46 × 8 × 11	1	0.98	0.78	0.74	0.74
Palm opening and closing	85 × 75 × 14 × 6	1	1.01	0.95	0.95	0.94

3-D Euclidean distance between the warped query video and the warped target video found using DTSWEF. The error is then normalized by dividing the error with the 3-D Euclidean distance of the warped query video and the warped target video found using DTSW.

Table 4.2: The normalized error of DTSWEF in comparing the karate punch, heart valve, and palm opening and closing videos with the variance = {100%, 90%, 80%, 70%}

Video set	Dimension	Normalized error of DTSW	Normalized error of DTSWEF			
			var=100	var=90	var=80	var=70
Karate punch	41 × 45 × 12 × 7	0	0	-0.036	0.001	0.036
Heart valve	40 × 46 × 8 × 11	0	0	1.8e-3	2e-3	0.024
Palm opening and closing	85 × 75 × 14 × 6	0	0	-0.013	-7e-3	0.021

The errors of DTSWEF in comparing the videos are less than 5% for the variance of at least 70%. For palm closing and opening video, the errors are negative for the variances of 90 and 80 percent. Similarly for the karate punch video with the variance of 90%. This means that using the reduced representation of the query video, the warped target video and the warped query video may become more similar than using the full representation of the query video. For karate punch and heart valve videos, we can achieve a 36% reduction in the execution time with less than 4% error in the results. For palm opening and closing video, we can achieve 6% reduction in the execution time with less than 3% error.

Figure 4-3 to 4-5 show the normalized execution time and normalized error for comparing the karate punch, heart valve, and palm opening and closing videos by

using Multiscale DTSWEF. We set the variance to be 10%, 20%, 30%, ..., 100%. In these experiments, we want to achieve at most 10% error in the results of Multiscale DTSWEF. The corresponding variance and the normalized execution time are shown. For the karate punch and palm opening and closing videos, we need at least 60% variance. For the heart valve videos, we can use Eigenframes that capture 10% variance of the original.

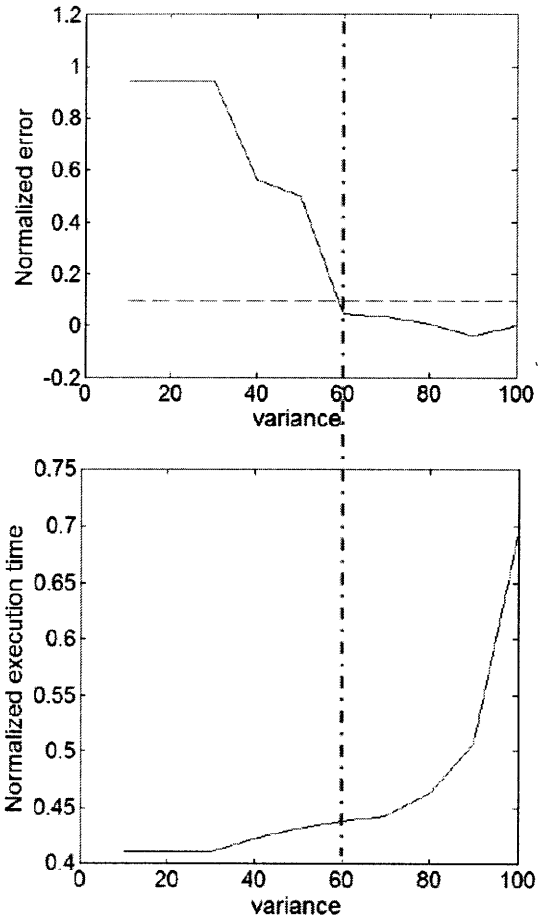

Figure 4-3: The normalized error and normalized execution time in comparing the karate punch videos using Multiscale DTSW with Eigenframes implementation and various settings for the variance

Table 4.3 summarizes the normalized execution time and the normalized error of DTSW, Multiscale DTSW, and Multiscale DTSWEF comparing the three example videos with the constraint that the normalized error must be less than 10%. Because the error and the execution time are normalized with the error and the execution

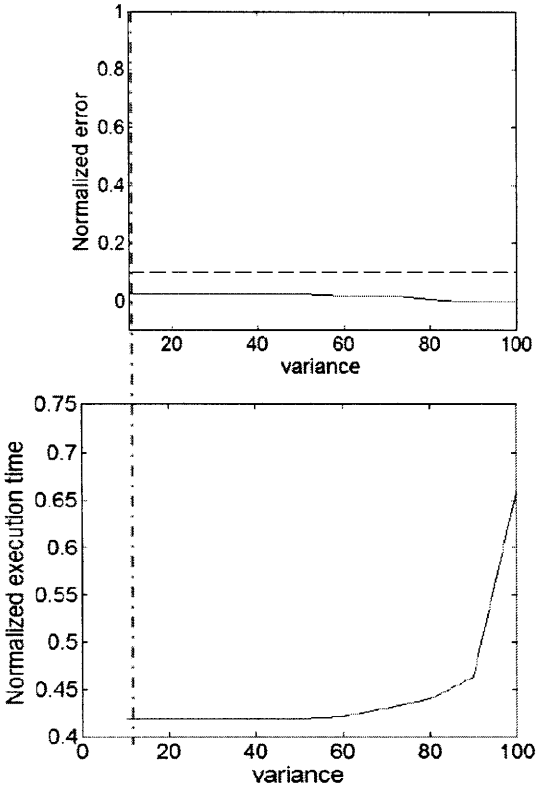

Figure 4-4: The normalized error and normalized execution time in comparing the heart valve videos using Multiscale DTSW with Eigenframes implementation and various settings for the variance

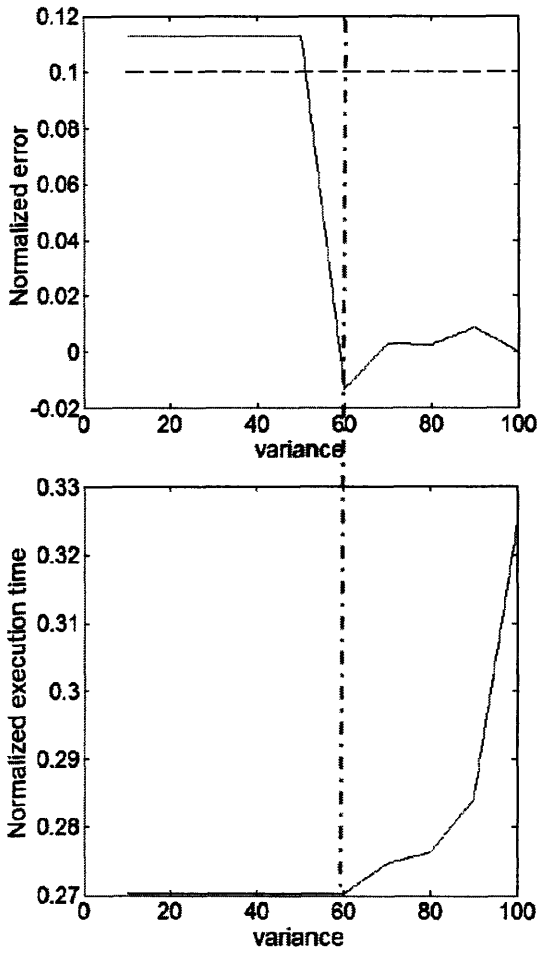

Figure 4-5: The normalized error and normalized execution time in comparing the palm opening and closing videos using Multiscale DTSW with Eigenframes implementation and various settings for the variance

time of DTSW, the normalized execution time and normalized error of DTSW are 1 and 0, respectively. For Multiscale DTSWEF, we chose the least amount of variance that still meet the requirement of maximum 10% error.

Table 4.3: The normalized execution time and the normalized error of DTSW, Multiscale DTSW, and Multiscale DTSWEF in comparing the karate punch, heart valve, and palm opening and closing videos with the constraint that the normalized error must be less than 10%

Video set	Dimension	DTSW		Multiscale DTSW		Multiscale DTSWEF	
		Normalized error	Normalized execution time	Normalized error	Normalized execution time	Normalized error	Normalized execution time
Karate punch	41 × 45 × 12 × 7	0	1	0	0.5252	0.0482	0.4383
Heart valve	40 × 46 × 8 × 11	0	1	0	0.534	0.0242	0.4149
Palm opening and closing	85 × 75 × 14 × 6	0	1	0	0.2878	0.0132	0.27

As shown in the table, the execution time of Multiscale DTSWEF is less than of Multiscale DTSW, and subsequently, the execution time of Multiscale DTSW is less than of DTSW. For karate punch and heart valve videos, the execution time of Multiscale DTSW is about 50% of the execution time of DTSW, and the execution time of Multiscale DTSWEF is about 40% of the execution time of DTSW. For the palm opening and closing video, Multiscale DTSW only needs 29% of the DTSW's execution time and Multiscale DTSWEF only needs 27% of the DTSW's execution time. The computational saving on large x, y, and time dimensions videos is more than the computational saving on smaller videos for Multiscale DTSW and Multiscale DTSWEF. This is because the multiscale approach saves computations in the time and space dimensions. And as explained in the previous chapter, the larger the dimension of the videos, the more computational saving is achieved by using the multiscale approach.

In conclusion, by using Multiscale DTSWEF, the computation of Multiscale DTSW is further reduced without greatly compromising the results. Due to relatively small number of computations that Multiscale DTSWEF needs, Multiscale DTSWEF is suitable for video classification, whose details are discussed in the next chapters.

4.2 Multiscale DTSW with Control Points

Suppose that we know a few points in the *Elemental Distance* and *Cumulative Distance* hypervolumes through which the optimal warp path must pass through. For instance, we know exactly which frames are the transitional frames between two different scenes in the target and query videos. A transitional frame in the target video must match the corresponding transitional frame in the query video. Therefore, we can assure that the points in the hypervolumes that represent the transitional frames in the target and query videos must be part of the sequence of points of the optimal warp path. We call such known information control points.

Then, based on boundary, temporal continuity, and bi-temporal casuality restrictions of DTSW, we predict the optimal warp path of the two videos in the time dimension. An example is shown in Figure 4-6 . In order to obtain a predicted optimal warp path in the space dimension, we follow the spatial continuity restriction of DTSW (the spatial change from frame to frame is bounded) and apply linear interpolation among the control points in the space dimension as illustrated in Figure 4-7. To get a more accurate solution, Multiscale DTSW will also compute the cells that are located at r_t and r_s cells away from the predicted path or cells in the computations of the hypervolumes.

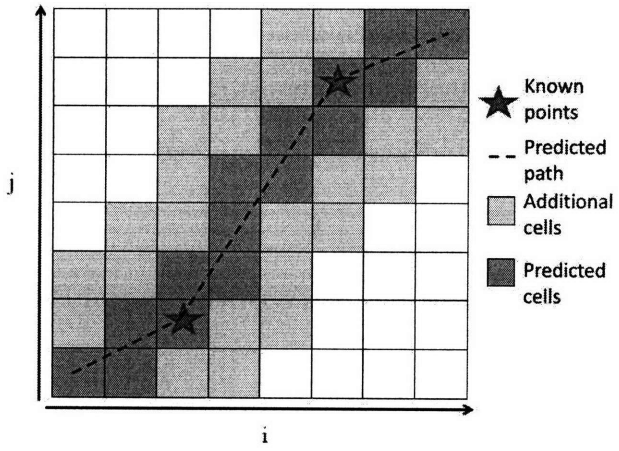

Figure 4-6: Known-points-based prediction of a warp path in the time dimension

We have implemented a modification of Multiscale DTSW that uses the predicted

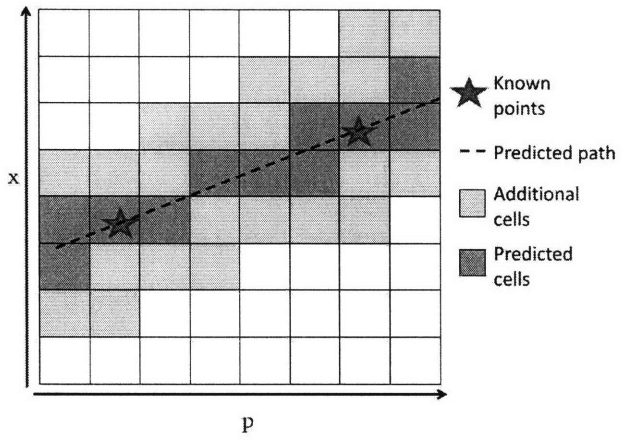

Figure 4-7: Known-points-based prediction of a warp path in the space dimension

optimal warp path (based on control points) to estimate the optimal warp path at the coarsest resolution. We called the modified Multiscale DTSW as Multiscale DTSW with Control Points. The cells in the hypervolumes that are considered as the candidates for the cells of the optimal warp path are called the predicted cells. Because we have an estimation of where the optimal warp path lies, Multiscale DTSW with Control Points will only search for the optimal warp path on the predicted cells at the coarsest resolution hypervolumes. Compared to Multiscale DTSW, which searches the optimal warp path on all cells of the coarsest resolution hypervolumes, Multiscale DTSW with Control Points has less computational cost.

The time complexity of Multiscale DTSW with Control Points is:

$$\begin{aligned} Cost(n) = 2(4r_t+3)(4r_s+3)^2 N \left(\sum_{i=0}^{i=n-1} \frac{1}{2^i} \right) + \\ 2b_x b_y N \left(\sum_{i=0}^{i=n-1} \frac{1}{2^i} \right) + 6N \left(\sum_{i=0}^{i=n-2} \frac{1}{2^i} \right). \end{aligned} \tag{4.1}$$

Compare it with Equation 3.21 of Multiscale DTSW, the number of computations

saved if we use control points is

$$\begin{aligned} Saving &= 2\left(\frac{N}{2^{n-1}}\right)^4 - \frac{N}{2^{n-1}}(2(4r_t+3)(4r_s+3)^2) \\ &= \frac{2}{2^{n-1}}\left(\frac{N^4}{2^{3(n-1)}} - N(4r_t+3)(4r_s+3)^2\right). \end{aligned} \tag{4.2}$$

Table 4.4 shows the computational cost of Multiscale DTSW and Multiscale DTSW with Control Points for comparing the karate punch videos (Appendix A). Both achieve the same optimal warp path at the finest resolution, but Multiscale DTSW with Control Points executed faster than Multiscale DTSW.

Table 4.4: Comparison of execution time of Multiscale DTSW and Multiscale DTSW with Control Points running on a Pentium M laptop with 1.3 GHz processor and 768 MB memory in comparing the karate punch videos

Videos set	Dimension	Execution time in seconds	
		Multiscale DTSW	Multiscale DTSW with Control Points
target1	41 × 45 × 12 × 7	175.62	172.45
target2	81 × 89 × 26 × 13	749.35	674.9
target3	81 × 89 × 52 × 28	7393.41	6855.46
target4	162 × 177 × 52 × 28	17572.48	14180.62

4.3 Multiscale DTSW with Level Jump

Figure 4-8 shows the optimal warp path for comparing the karate punch videos (Appendix A) at each level of the execution of Multiscale DTSW. Figure 4-9 depicts the projection of the optimal warp path at each level to a common time axis. We can observe that for some levels (level 2 and 4), the optimal warp path found is almost the same as the optimal warp path found at its lower level (The lowest level is for the coarsest resolution and the highest level is for the finest resolution). Therefore, Multiscale DTSW can skip certain levels and project the optimal warp path to two or three times finer resolution.

To find the similarity between two optimal warp paths from two different resolutions, we must first project both optimal warp paths to a common axis. Then, the

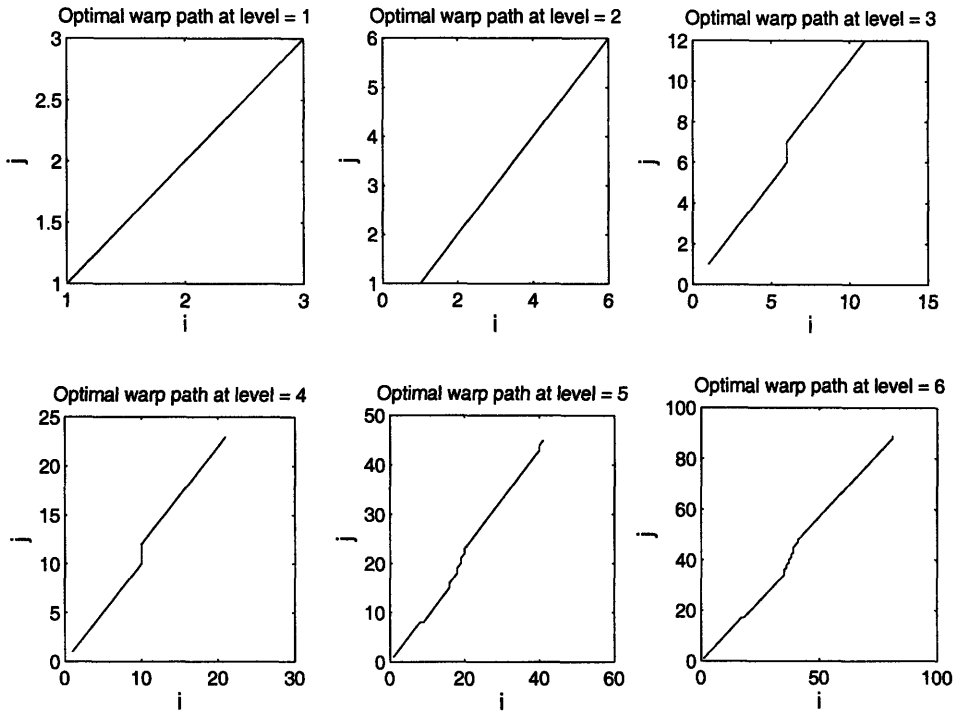

Figure 4-8: The optimal warp paths of comparing the karate punch videos found by using Multiscale DTSW at level = 1 to level = 6

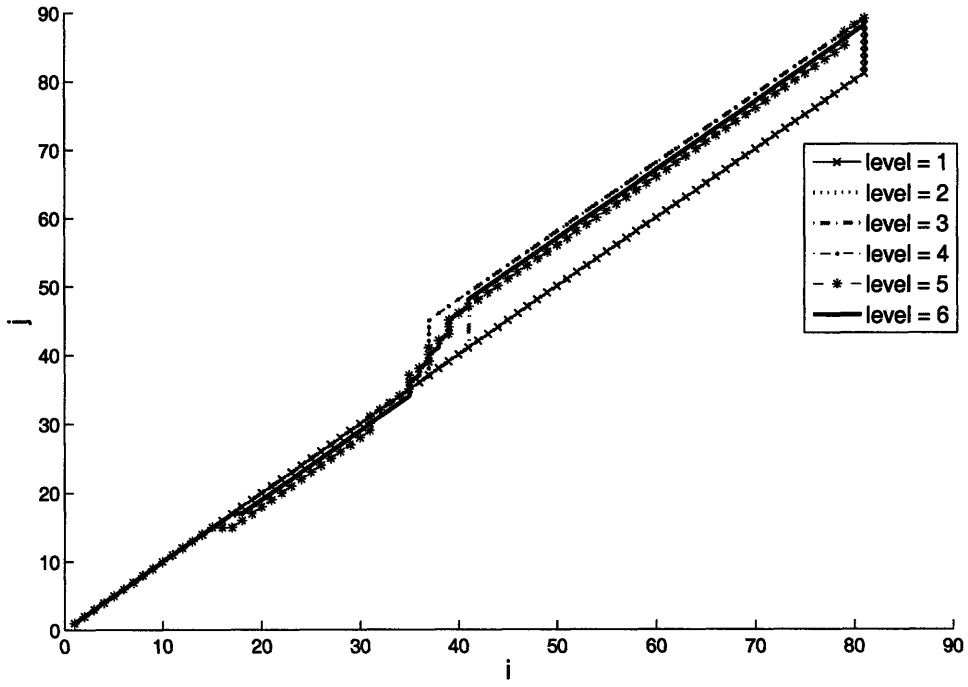

Figure 4-9: Projection of all optimal warp pathes of comparing the karate punch videos found by using Multiscale DTSW at level=1 to level=6 onto a common axis

difference between the optimal warp paths is computed by the following equation.

$$Total\_Warp\_Diff = \sum_{i=1}^{i=N} \left(\frac{Wq_1(i) - Wq_2(i)}{N} \right), \tag{4.3}$$

where N is the length of the time dimension of the common axis. And Wq_1 and Wq_2 are the $W_k(j)$s of both optimal warp paths projected to a common axis, as shown in Figure 4-10. In other words, the difference between two optimal warp paths is defined as the summation of the absolute vertical differences between both optimal warp paths projected to a common axis.

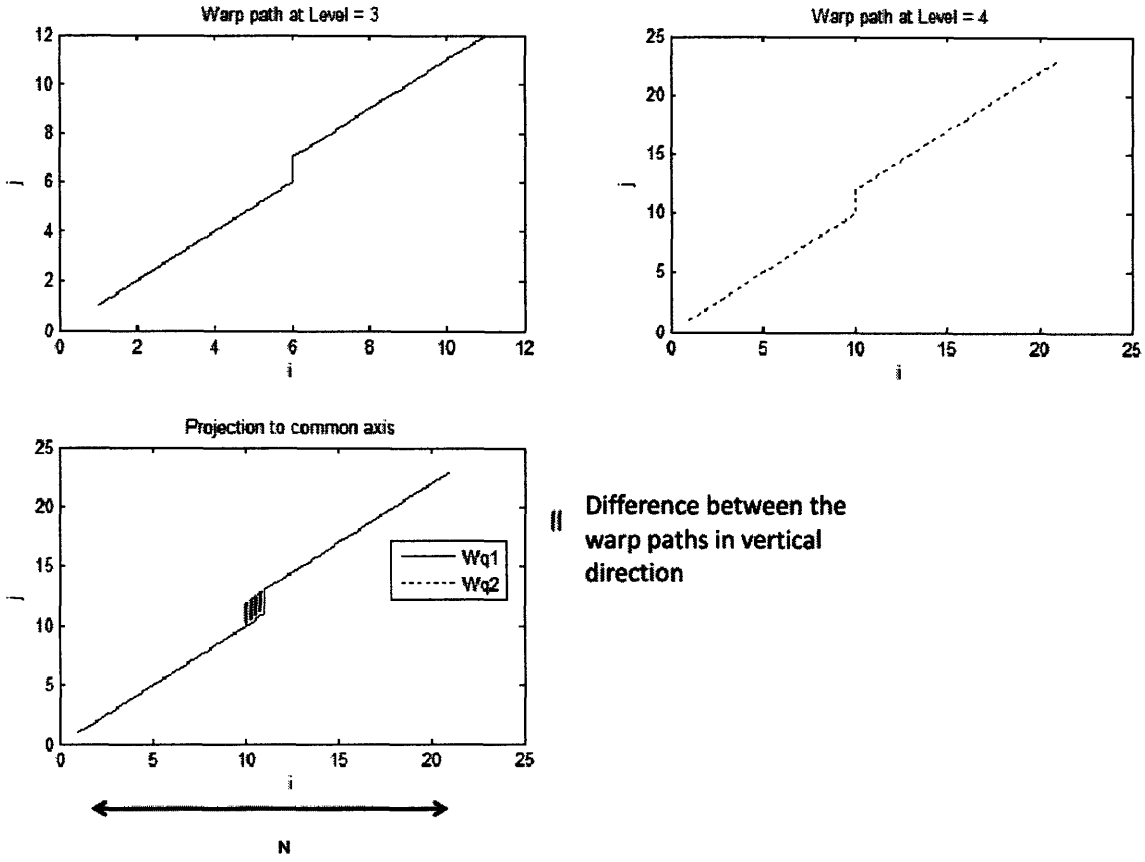

Figure 4-10: The top left figure shows an example of the optimal warp path at level=3. The top right figure shows an example of the optimal warp path at level=4. The bottom figure shows the projection of both optimal warp paths to a common axis. The difference between the two optimal warp paths is defined as the summation of absolute vertical differences between both warp paths projected to a common axis as shown in the shaded region in the figure.

We have implemented a modification of Multiscale DTSW that will skip some

levels of resolution if Multiscale DTSW has detected that the optimal warp path at some resolution and the optimal path at the finer resolution do not have substantial change. We call the modified Multiscale DTSW as Multiscale DTSW with Level Jump.

Figure 4-11 shows the flowchart of Multiscale DTSW with Level Jump. We define two threshold, *threshold*1 and *threshold*2. If $Total\_Warp\_Diff$ is smaller than *threshold*1, then Multiscale DTSW with Level Jump skips two levels. Otherwise, if $Total\_Warp\_Diff$ is smaller than *threshold*2, then Multiscale DTSW with Level Jump will skip one level. Other than that, Multiscale DTSW with Level Jump will continue to the next level.

Table 4.5 shows the execution time of DTSW, Multiscale DTSW, and Multiscale DTSW with Level Jump in comparing the karate punch videos. All achieve the same optimal warp path at the finest resolution, but Multiscale DTSW with Level Jump executed faster than Multiscale DTSW.

Table 4.5: Comparison of execution time of DTSW, Multiscale DTSW, and Multiscale DTSW with Level Jump in comparing the karate punch videos running on a Pentium M laptop with 1.3 GHz processor and 768 MB memory

Videos set	Dimension	Execution time in seconds		
		DTSW	Multiscale DTSW	Multiscale DTSW with Level Jump
target1	$41 \times 45 \times 12 \times 7$	283.2	177.32	96.57
target2	$81 \times 89 \times 26 \times 13$	7616.24	1794.87	1122.47

Multiscale DTSW with Level Jump can also be extended to use more than two thresholds. However, because the value of the optimal level is relatively small, it is quite unlikely that the algorithm is able to skip more than two levels.

4.4 Piece-wise Multiscale DTSW

To improve the result of Multiscale DTSW, we can divide a query and a target video into several parts and then apply Multiscale DTSW on each portion independently. After we find the optimal warp path, we warp each portion of the target video according to the optimal warp path. We then combine each of these portions in common

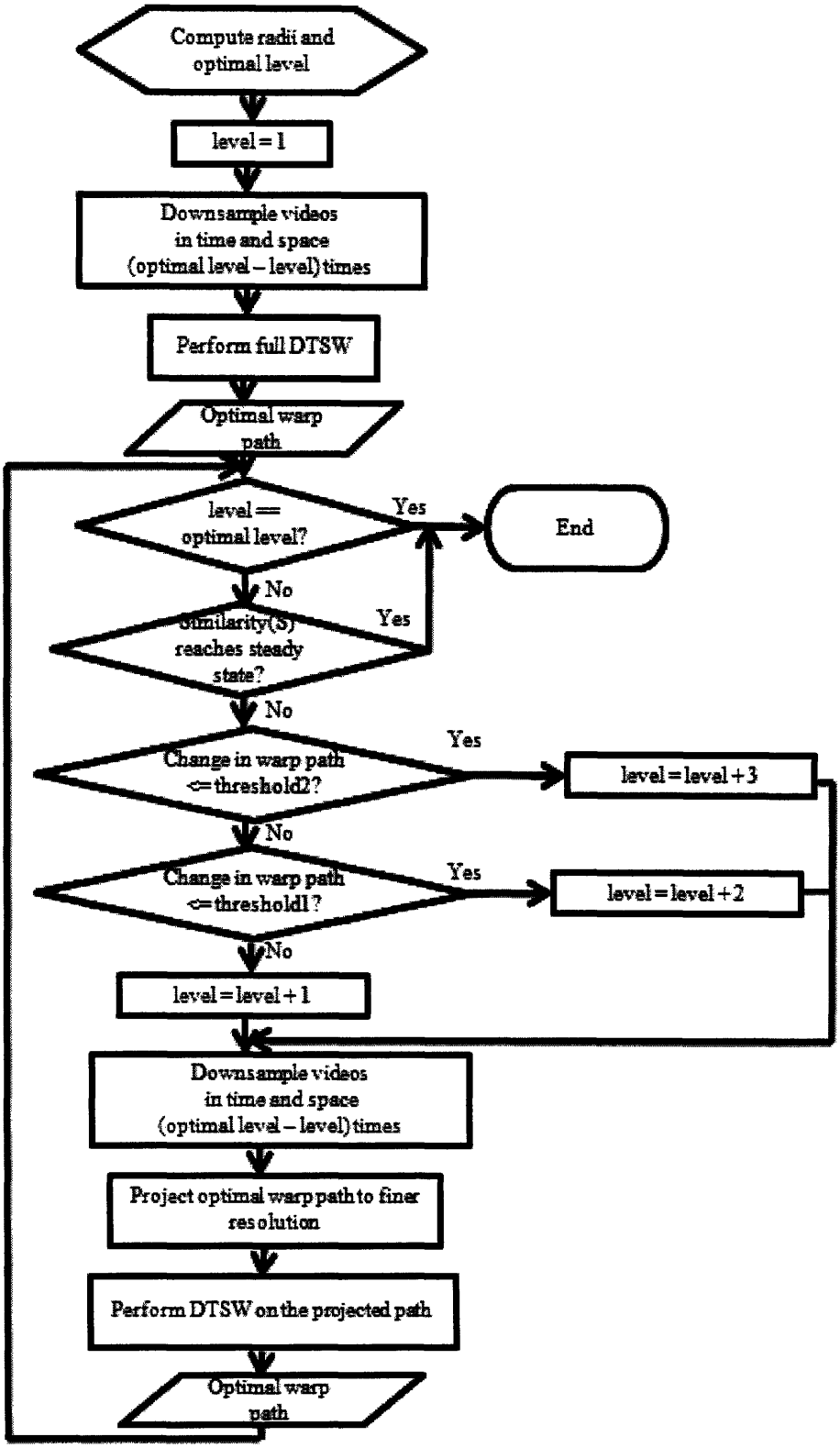

Figure 4-11: Flowchart of Multiscale DTSW with Level Jump

time and space axes. We call this method as Piece-wise Multiscale DTSW. By using Piece-wise Multiscale DTSW, the warped target video is more similar to the query video. However, Piece-wise Multiscale DTSW needs more computations. More detailed explanation about Piece-wise Multiscale DTSW including the computational cost and the warped target video can be found in Appendix B.

Chapter 5

Multiscale DTSW in Video Classification Application

5.1 Video Classification Application

Because of its efficiency, Multiscale DTSW is applicable to video classification application. In this research, we developed a video classification application that is based on Multiscale DTSW. The video classification application classifies an unknown video into one of the predefined classes: walk, skip, side, run, and jump. This video classification application determines if the action performed by the person in the unknown video is a walk, skip, side, run, or jump action. We assume that the unknown video represents one of these five actions, the application did not implement an exception in case of the action in the unknown video does not belong to one of the five actions. However, this could be easily done by setting a threshold. If the distance or difference between the unknown video and all the videos in the database exceeds the threshold, the classification result will show that the unknown video does not belong to one of the classes.

In this research, we have four template videos for each class. Therefore, the total number of videos in the database is 20 videos. The examples of the template videos can be found in Appendix A. As most video classification applications [3, 20, 28], we use the nearest neighbor procedure in determining the classification result. Figure 5-1

shows the steps in the video classification application. In the first step, the application finds the distance or difference between the unknown video and each of the template videos in the database. To find the distance, we will use DTSW, Multiscale DTSW or Multiscale DTSWEF. In the second step, the application computes the average of the distance between the unknown video and the template videos of each class. In the final step, the application finds the minimum of the average distance among all the classes. The class that has the minimum average distance between the unknown video and each of its template video is the output class.

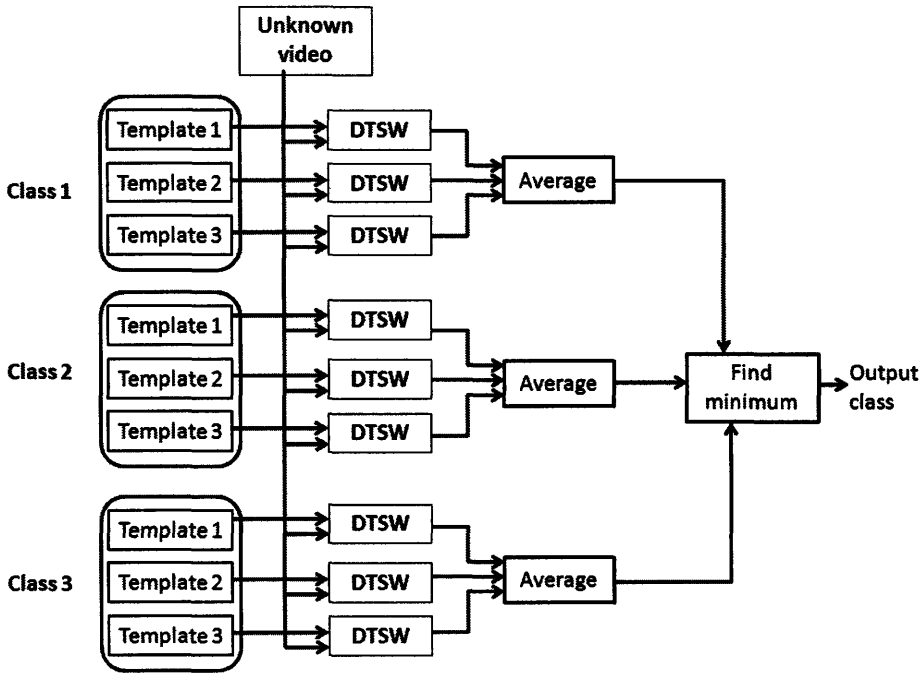

Figure 5-1: The diagram of the video classification application

In this research, we have 25 unknown videos, each five of which are the videos showing a person performing an action that belongs to one of the five classes.

5.2 Selection of Decision Variables

There are many decision variables such as resolution, b_x, and b_y that need to be set in the DTSW, Multiscale DTSW, or Multiscale DTSWEF algorithm. The settings of the variables depend on the type of the video that we are comparing. In this section,

we discuss how we set some of these variables for the video classification application.

We want to set the value for each decision variable that best represents the type of the videos that we are comparing and improves the performance of the video classification application. We call the set of template videos in the database as the template set. The video from which a template video is extracted is called the template source video. Figure 5-2 illustrates the template source video. The set of the template source videos is called the training set. To determine the value of each decision variable, we performed some experiments on the training set.

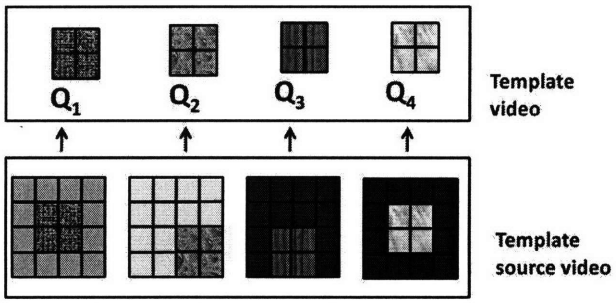

Figure 5-2: An example of a template video and the corresponding template source video

We run the classification on the training set and find the values of the decision variables that classify the training set best. To find the best setting, we plot the Receiver Operator Characteristic (ROC) of the classifier with different settings of the decision variables.

The ROC plot is the plot of False Positive Rate (FPR) versus True Positive Rate (TPR). It is usually applied for classification with two classes: positive or negative. Figure 5-3 shows an example of the ROC plot. The FPR ranges from 0 to 1 and it is calculated by

$$FPR = \frac{|FP|}{|FP| + |TN|}. \tag{5.1}$$

False Positive (FP) means that the predicted value is positive but the actual value is negative. True Negative (TN) means that the predicted and the actual value are

negative. The TPR also ranges from 0 to 1 and it is computed by

$$TPR = \frac{|TP|}{|TP| + |FN|}. \tag{5.2}$$

True Positive (TP) means that the predicted and the actual value are positive. False Negative (FN) means that the predicted value is negative but the actual value is positive.

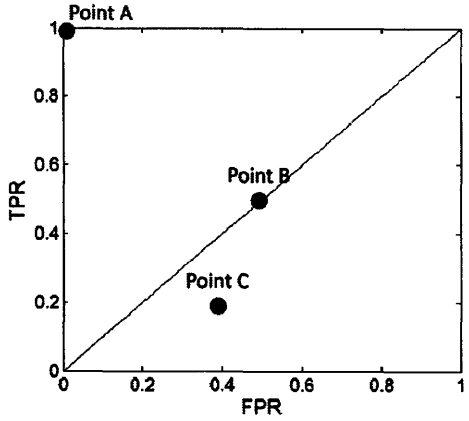

Figure 5-3: An example of Receiver Operator Characteristic (ROC) plot

The best classifier will be the classifier whose ROC plot is at the top left corner (point A in Figure 5-3); the FPR is zero and the TPR is one. If the plot lies below the diagonal line (like point C in Figure 5-3), then the classification result is most probably wrong. If the plot lies along the diagonal line (like point B in Figure 5-3), then the classification has 50-50 percent chance of getting the correct result. The further the plot lies above the diagonal line, the better the classifier is.

To plot the ROC for five classes classification, we compute the average FPR and TPR of the classification between one class and the union of the rest of the classes. Specifically, we first compute the FPR and TPR of the classification between class 1 and the combined class 2, 3, 4, and 5. Then, we compute the FPR and TPR of the classification between class 2 and the combined class 1, 3, 4, and 5. We repeat this computation for all the classes and then take the average to find the FPR and TPR for our classification application.

The accuracy of the classifier is defined as:

$$Accuracy = \frac{|TP| + |TN|}{|FP| + |FN| + |TP| + |TN|}. \tag{5.3}$$

The accuracy ranges from 0 to 1.

In this thesis, we discuss four decision variables: motion scalar combination, resolution, b_x and b_y, and variance for Multiscale DTSWEF.

5.2.1 Motion Scalar Combination

Because our application classifies an action (i.e. motion), it is appropriate to use motion scalar combination as the basic data of the video instead of the intensity or histogram. There are two components in the optical flow: movement in the x direction or **u**, and movement in the y direction or **v**. The **u** and **v** of the template video are computed from the motion in the template source video. From these two components, we calculate six different pixel indexed scalar combinations of motion: **u**, **v**, $\mathbf{u}^2$, $\mathbf{v}^2$, **uv**, and $\mathbf{u}^2\mathbf{v}^2$.

To choose the motion scalar combination, we fixed the resolution of the template video at $21 \times 3 \times 4$ (x × y × time), and $b_x = b_y = 10$. Figure 5-4 shows the ROC plot of the video classification application on the training set for different motion scalar combinations. Figure 5-5 shows the ROC plot of the video classification application on the unknown set for different motion scalar combinations. The unknown set is the set of unknown videos. Table 5.1 shows the accuracy of the classification result for each motion scalar combination on the training and unknown sets.

Table 5.1: The accuracy of the video classification application on the training and unknown sets for different motion scalar combinations

Motion scalar combination	Accuracy (training set)	Accuracy (unknown set)
u	0.8	0.76
v	0.65	0.8
$\mathbf{u}^2$	0.55	0.68
$\mathbf{v}^2$	0.55	0.52
uv	0.5	0.44
$\mathbf{u}^2\mathbf{v}^2$	0.25	0.28

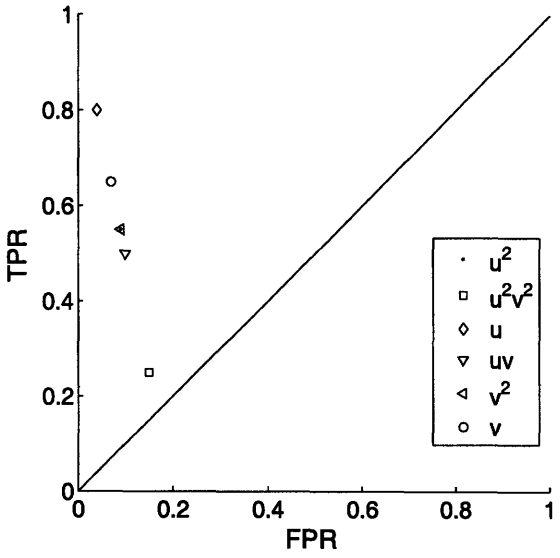

Figure 5-4: ROC plot of the video classification application on the training set with different motion scalar combinations

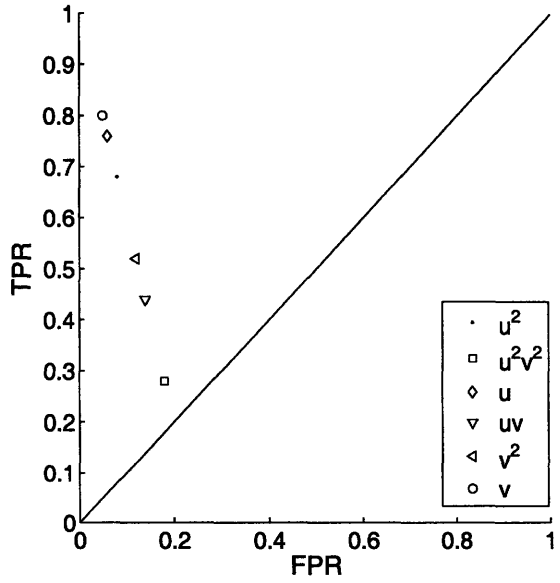

Figure 5-5: ROC plot of the video classification application on the unknown set with different motion scalar combinations

From Figure 5-4 we can see that $\mathbf{u}^2\mathbf{v}^2$ is a bad choice for the motion scalar combination. We can also see that $\mathbf{u}$ is the best motion scalar combination to classify the training set. This is further supported by the accuracy data in Table 5.1. The accuracy of classifying the training set by using $\mathbf{u}$ as the motion scalar combination is 80%. To check if the $\mathbf{u}$ scalar combination can classify the unknown set as well, we also performed the classification on the unknown set using different settings of the motion scalar combination. Based on Figure 5-5 and Table 5.1, the $\mathbf{u}$ motion scalar combination is the second best scalar combination to classify the unknown set. Therefore, it is still acceptable to use the $\mathbf{u}$ motion scalar combination in our classification application.

There is also a variation to the $\mathbf{u}$ and $\mathbf{v}$ components of the optical flow: the **normalized u** and **normalized v** components. We call these components: u_N and v_N. u_N and v_N are the $\mathbf{u}$ and $\mathbf{v}$ components that are normalized such that the length of vector

$$\begin{pmatrix} u_N \\ v_N \end{pmatrix}$$

is equal to 1. For this option, we also have six different normalized motion scalar combinations: u_N, v_N, ${u_N}^2$, ${v_N}^2$, $u_N v_N$, ${u_N}^2{v_N}^2$.

Figure 5-6 shows the ROC plot of the video classification application on the training set with different settings of the normalized motion scalar combination. Figure 5-7 shows the ROC plot of the video classification application on the unknown set with different settings of the normalized motion scalar combination. Table 5.2 shows the accuracy of the classification result for each normalized motion scalar combination on the training and unknown sets.

From Figure 5-6 we can see that ${u_N}^2{v_N}^2$ is a bad choice for the normalized motion scalar combination. We can also see that ${u_N}^2$ is the best motion scalar combination to classify the training set. This is further supported by the accuracy data in Table 5.2. The accuracy of classifying the training set by using ${u_N}^2$ as the motion scalar combination is 85%. To check if the ${u_N}^2$ scalar combination can classify the unknown set as well, we also performed the classification on the unknown set using

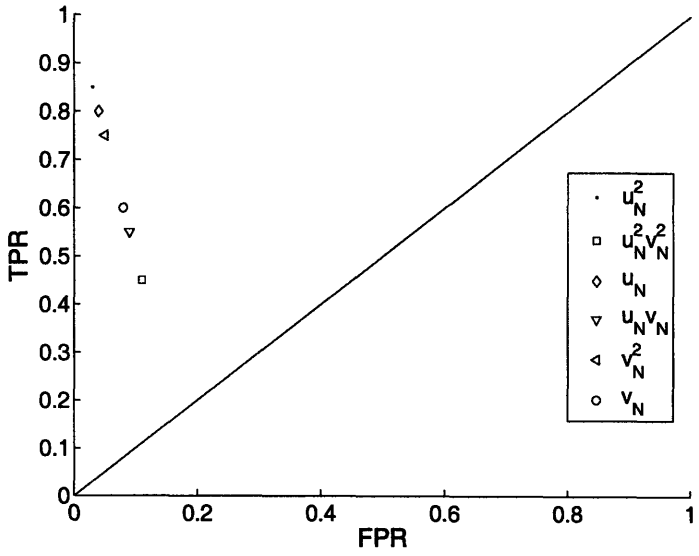

Figure 5-6: ROC plot of the video classification application on the training set with different settings of the normalized motion scalar combination

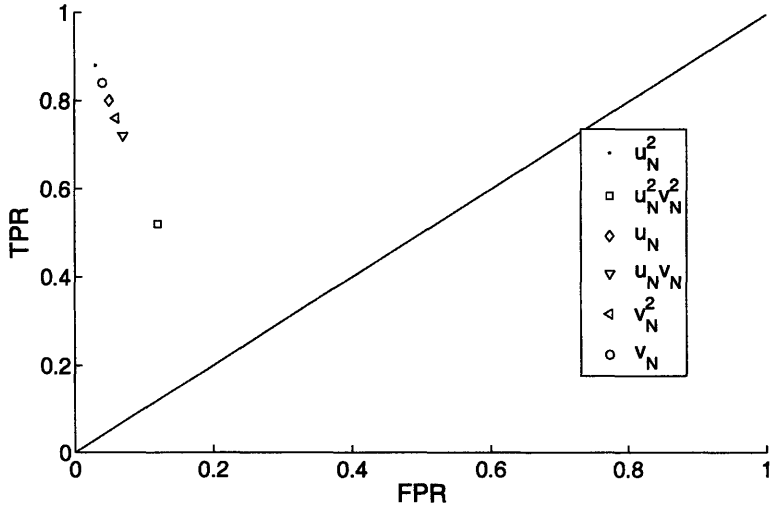

Figure 5-7: ROC plot of the video classification application on the unknown set with different settings of the normalized motion scalar combination

Table 5.2: The accuracy of the video classification application on the training and unknown sets for various settings of the normalized motion scalar combination

Normalized motion scalar combination	Accuracy (training set)	Accuracy (unknown set)
u_N	0.8	0.8
v_N	0.6	0.84
${u_N}^2$	0.85	0.88
${v_N}^2$	0.75	0.76
$u_N v_N$	0.55	0.72
${u_N}^2{v_N}^2$	0.45	0.52

different settings of the normalized motion scalar combination. Based on Figure 5-6 and Table 5.2, the ${u_N}^2$ scalar combination is also the best component to classify the unknown set. Therefore, we should use ${u_N}^2$ scalar combination in the video classification application if we want to use the normalized motion scalar combination instead of the motion scalar combination.

In summary, we select the normalized motion scalar combination by performing the classification on the training set and pick the best normalized motion scalar combination to classify the training set. We expect that the same scalar combination will classify the unknown set with a good accuracy.

5.2.2 Resolution

We want to downsample the unknown and template videos because it is faster to run the DTSW or Multiscale DTSW algorithm on smaller videos. However, we do not want to downsample so much that the performance of the classification application is degraded.

We downsample both the unknown and the template videos by a common factor in the time and space dimensions. To find the best resolution, we chose $\mathbf{u}$ as our motion scalar combination and ${u_N}^2$ as our normalized motion scalar combination based on the classification result on the training set. The b_x and b_y are fixed at 10. The resolution is represented by the length of x dimension $\times$ y dimension $\times$ time dimension.

Figure 5-8 shows the ROC plot of the classification result on the training set for

different resolutions with ${u_N}^2$ as the motion scalar combination. Figure 5-9 shows the ROC plot of the classification result on the unknown set for different resolutions with ${u_N}^2$ as the motion scalar combination.

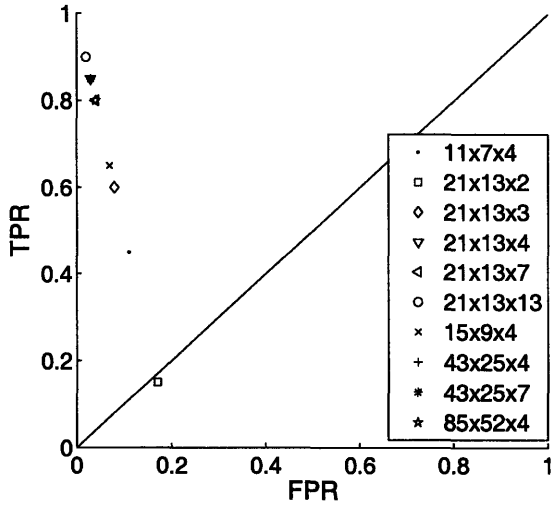

Figure 5-8: ROC plot of the video classification application on the training set using ${u_N}^2$ as the motion scalar combination and different sets of resolution

Table 5.3 shows the accuracy of the classification result on the training and unknown sets.

Table 5.3: The accuracy of the video classification application on the training and unknown sets for various sets of resolution with ${u_N}^2$ as the motion scalar combination

Resolution	Accuracy (training set)	Accuracy (unknown set)
$11 \times 7 \times 4$	0.45	0.44
$21 \times 13 \times 2$	0.15	0.16
$21 \times 13 \times 3$	0.6	0.4
$21 \times 13 \times 4$	0.85	0.88
$21 \times 13 \times 7$	0.8	0.92
$21 \times 13 \times 13$	0.9	0.88
$15 \times 9 \times 4$	0.65	0.6
$43 \times 25 \times 4$	0.85	0.76
$43 \times 25 \times 7$	0.85	0.92
$85 \times 51 \times 4$	0.8	0.72

As seen from Figure 5-8 and Table 5.3, the best resolution for the classification on the training set with ${u_N}^2$ as the motion scalar combination is $21 \times 13 \times 13$. In the

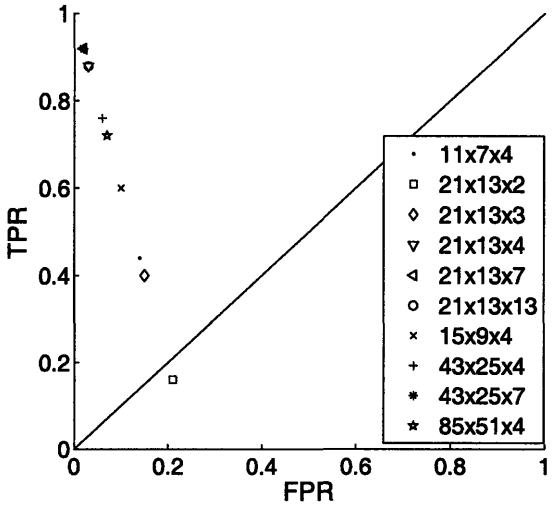

Figure 5-9: ROC plot of the video classification application on the unknown set using ${u_N}^2$ as the motion scalar combination and different sets of resolution

unknown set, this resolution produces the second best classifier. The best classifier classifies the unknown set with 92% accuracy, and the classifier with the resolution of $21 \times 13 \times 13$ classifies the unknown set with the accuracy of 88%.

Figure 5-10 shows the ROC plot of the classification result on the training set for different resolutions with $\mathbf{u}$ as the motion scalar combination. Figure 5-11 shows the ROC plot of the classification result on the unknown set for different resolutions with $\mathbf{u}$ as the motion scalar combination.

Table 5.4 shows the accuracy of the classification result on the training and unknown sets.

As seen from Figure 5-10 and Table 5.4, the best resolution for the classification with $\mathbf{u}$ as the motion scalar combination is $21 \times 13 \times 13$. In the unknown set, this resolution also produces the best classifier. In the training and unknown sets, both classification results are of 100% accuracy. In summary, we do not need the full resolution which is $85 \times 51 \times 13$, but with smaller resolution ($21 \times 13 \times 13$), we have a 100% accurate classification.

After this stage, we chose the motion scalar combination to be $\mathbf{u}$ and the resolution to be $21 \times 13 \times 13$.

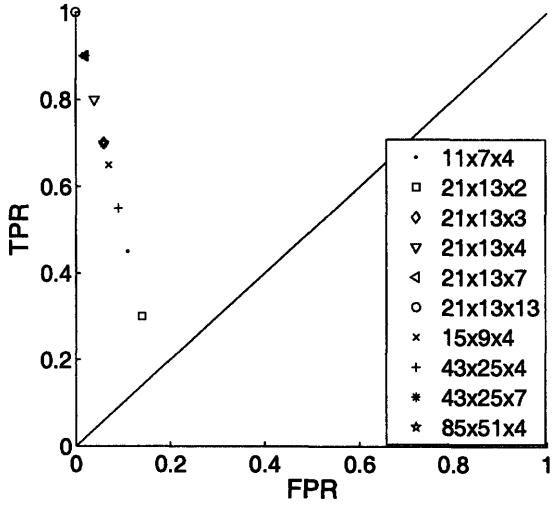

Figure 5-10: ROC plot of the video classification application on the training set using **u** as the motion scalar combination and different sets of resolution

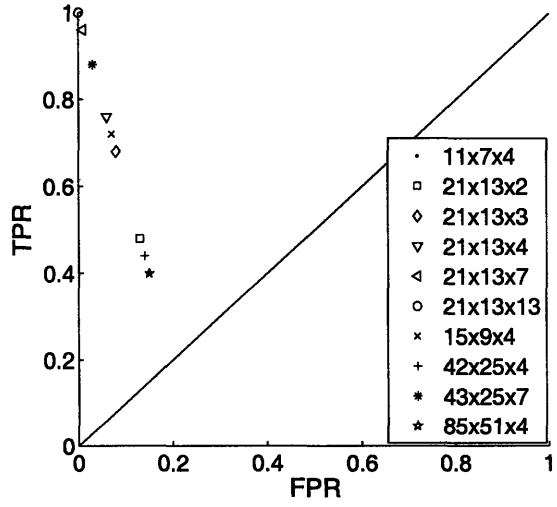

Figure 5-11: ROC plot of the video classification application on the unknown set using **u** as the motion scalar combination and different sets of resolution

Table 5.4: The accuracy of the video classification application on the training and unknown sets for various sets of resolution with **u** as the motion scalar combination

Resolution	Accuracy (training set)	Accuracy (unknown set)
$11 \times 7 \times 4$	0.45	0.4
$21 \times 13 \times 2$	0.3	0.48
$21 \times 13 \times 3$	0.7	0.68
$21 \times 13 \times 4$	0.8	0.76
$21 \times 13 \times 7$	0.9	0.96
$21 \times 13 \times 13$	1	1
$15 \times 9 \times 4$	0.65	0.72
$43 \times 25 \times 4$	0.55	0.44
$43 \times 25 \times 7$	0.9	0.88
$85 \times 51 \times 4$	0.7	0.9

5.2.3 Maximum Allowable Changes in the x and y Dimensions

Variables b_x and b_y determine the maximum allowable changes from a frame to the next frame in the x and y dimensions of the optimal warp path. The bigger the values of b_x and b_y are, the further we allow the space shifting in the optimal warp path. The bigger the values of b_x and b_y also means that more computations required by Multiscale DTSW since the total computation of Mutltiscale DTSW is $4N(4r_t + 3)(4r_s + 3)^2 + (4N \times b_x \times b_y) + 12N$. Therefore, we will try to keep the value of b_x and b_y as small as possible without affecting the accuracy of the classification result.

Until this stage, we fix $b_x = b_y = 10$, and we are able to get a 100% accurate classification result. So, we will gradually reduce the value of b_x and b_y until the accuracy of the classification result is less than 100%. Table 5.5 shows the accuracy of the classification result on the training set with the values of $b_x = b_y = \{10, 8, 6, 4, 2, 1\}$. As can be observed from the table, the smallest value for b_x and b_y that still produces a 100% accurate classification result is two. Hence, we set the b_x and b_y to be two.

Up to this stage, we have set all the necessary decision variables for the video classification application that bases on DTSW or Multiscale DTSW. We set the motion scalar combination to be **u**, the resolution to be $21 \times 13 \times 13$, and $b_x = b_y = 2$. Either

Table 5.5: The accuracy of the video classification application on the training set for various values of b_x and b_y

Value of $b_x = b_y$	Accuracy (training set)
10	1
8	1
6	1
4	1
2	1
1	0.95

DTSW-based video classification or Multiscale-DTSW-based video classification has successfully classified the unknown set with a 100% accuracy with these settings. The execution time for both applications can be found in Subsection 5.3.1.

5.2.4 Variance for Multiscale DTSWEF

As discussed in the previous chapter, one way to reduce the complexity of Multiscale DTSW is by using Eigenframes representation. For the Eigenframes implementation, we need to set the variance. A lower variance threshold tends to result in fewer required Eigenframes and therefore less computations. However, the lower the variance, the more likely that the classification application's performance is degraded. We run the video classification application on the training set with different variance settings in order to select the appropriate variance's value.

Table 5.6 shows the accuracy of the Multiscale-DTSWEF-based video classification application on the training set with different variance settings. The table shows that the classification application's performance is degrading as we reduce the variance. If we want to have a 100% accurate solution, we should use a variance of 100%. But, if we allow a 10% error tolerance, we can use a variance of 90%.

Table 5.6: The accuracy of the Multiscale-DTSWEF-based video classification application on the training set with various settings for the variance

Variance	Accuracy (training set)
100%	1
90%	0.95
80%	0.85
70%	0.75

5.3 The Performance of the Video Classification Application

5.3.1 Comparison between DTSW, Multiscale DTSW, and Multiscale DTSWEF

Table 5.7 shows the normalized execution time and percentage error of the video classification application on the unknown set using DTSW, Multiscalse DTSW, and Multiscale DTSWEF. We chose to tolerate a 10% error in the classification result. Therefore, for the Multiscale-DTSWEF-based video classification application, we set the variance at 90% (based on the experiments in the previous subsection). The normalized execution time is the execution time divided by the execution time of DTSW-based video classification application.

Table 5.7: The normalized execution time and percentage error of the video classification application based on DTSW, Multiscale DTSW, and Multiscale DTSWEF on the unknown set

Video comparison algorithm	Normalized execution time	Percentage error
DTSW	1	0
Multiscale DTSW	0.475	0
Multiscale DTSWEF	0.458	0

From the table, we can observe that the Multiscale-DTSW-based video classification is much more efficient that DTSW-based video classification application, and Multiscale-DTSWEF-based video classification application is more efficient than Multiscale-DTSW-based video classification application. All applications achieve the same amount of accuracy but the Multiscale-DTSW-based application only required

less than half of the execution time of the DTSW-based application, and Multiscale-DTSWEF-based application only required about 46% of the execution time of the DTSW-based application.

Besides classifying walk, run, side, skip, and jump actions, we also have developed a video classification application to classify left-to-right, circular, and hop pointing actions. The illustration of the difference among the pointing actions and examples of the template videos can be found in Appendix A. We have a total of 12 template videos in the database for classifying these three classes.

For this video classification application, we did the same experiments as mentioned above to decide the best value for each of the decision variables. The best motion scalar combination is $\mathbf{v}$ with the resolution of $30 \times 40 \times 11$, and a value of 10 for b_x and b_y. Table 5.8 shows the accuracy of the video classification application on the training set for different settings of the variance for the Multiscale DTSWEF. From the table, we can observe that the accuracy of the video classification application on the training set for a variance less than 100% is quite low. Therefore, we decided not to use Multiscale DTSWEF in this video classification application.

Table 5.8: The accuracy of the Multiscale-DTSWEF-based video classification application on the training set (left-to-right, circular, and hop sets) for various settings for the variance

Variance	Accuracy (training set)
100%	0.9167
90%	0.8333
80%	0.667
70%	0.667

We run the DTSW-based and Multiscale-DTSW-based video classification applications on the unknown set. There are 15 unknown videos on the unknown set. Each five videos are the videos showing a person performing a pointing action that belongs to one of the three classes. Table 5.9 shows the normalized execution time and the accuracy of the applications in classifying the three classes. Both applications achieved a 100% accurate classification, but Multiscale-DTSW-based video classification application executed in 57% of the execution time of DTSW-based video classification

application.

Table 5.9: The normalized execution time and accuracy of the video classification application based on DTSW and Multiscale DTSW on the unknown set in classifying left-to-right, circular, and hop pointing actions

Video comparison algorithm	Normalized execution time	Accuracy
DTSW	1	1
Multiscale DTSW	0.568	1

5.3.2 Sensitivity to Scale Variance

We carried out experiments to investigate the sensitivity of our video classification to scale variation. To obtain different scales for the template video, we downsample or upsample the template video in the space dimension and keep the unknown video unchanged. By doing this, the size of the person jumping in the template video and the unknown video will be different.

Table 5.10 shows the result of the classification with different scales of the template video.

Table 5.10: The accuracy of the video classification application on the unknown set for various scale differences between the template and unknown videos

Size of template	Percentage size of template	Accuracy (unknown set)
15×9	71.43%	0.5
17×11	80.95%	0.8182
21×13	100%	1
31×10	147.62%	0.7727

If we tolerate 20% error in the classification result, then the classification application is insensitive to a variation of 20% in the scale difference.

Chapter 6

Multiscale DTSW in Video Detection Application

In this chapter, we explore Multiscale DTSW applied to video detection.

6.1 Video Detection Application

Figure 6-1 outlines the video detection application that we developed. In the first step, we find subvideos of the target video. The length of each subvideo is the same and the length may or may not be the same as the length of the query video. In the second step, we apply the video comparison algorithm to compare each subvideo to the query video. The video comparison algorithm is either Dynamic Space Warping (DSW) or Multiscale DTSW. More explanation about these methods or algorithms can be found in the next section. In the third step, we stack into a *Detection Volume* all the $\{x, y\}$ planes of the last temporal location of the optimal warp path of comparing each subvideo to the query video. In the fourth step, we filter the *Detection Volume* with a low-pass filter. We chose an averaging filter as our low-pass filter. In the final step, we compute the minimum distance of each $\{x, y\}$ plane in the filtered *Detection Volume* and plot it against the frame number of the first frame of the corresponding subvideo. The detected temporal location of the query video will be the local minimum in the plot. There may be more than one local minimum, and therefore, there may be

more than one detected temporal location. The detected spatial location of the query video will be the $\{x, y\}$ coordinate of the minimum distance in the $\{x, y\}$ plane of the detected query temporal location in the filtered *Detection Volume*.

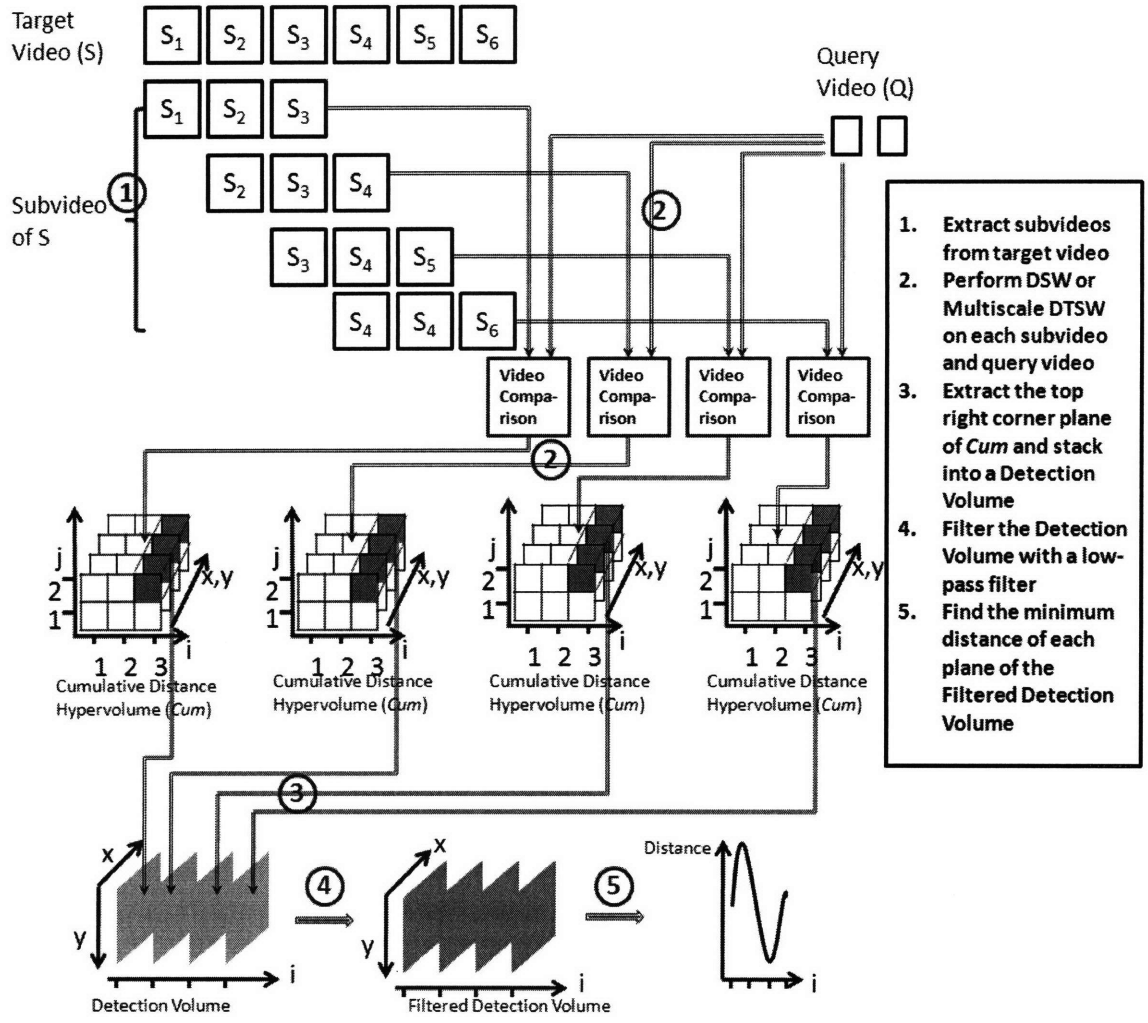

Figure 6-1: The diagram of the video detection application

6.2 Methods for Comparing Videos in the Video Detection Application

We have three different methods for the video comparison unit in the video detection application: Dynamic Space Warping (DSW), Multiscale DTSW, and a combination of DSW and Multiscale DTSW.

6.2.1 DSW

As discussed in Anthony [1], DTSW is simplified if we use a common time axis or ignore the temporal variation in the time dimension. We call this technique Dynamic Space Warping (DSW). DSW is similar to DTSW except that we do not warp the videos along the time dimension. Since, no warping in the time dimension, the optimal warp path in the time dimension will be a straight diagonal path. The first frame of the target video is matched to the first frame of the query video. However, we still allow a shifting or warping in the space dimension.

The number of computations of DSW is relatively small since we only fill in the *Elemental Distance (D)* and *Cumulative Distance (Cum)* hypervolumes at the diagonal path along the time dimension. Assume that I, J, X, and Y in the D and Cum hypervolumes are all equal to N, then the length of the diagonal path along the time dimension is N. Hence, DSW only fills in the D and Cum hypervolumes N cells in the time dimension for each cell in the spatial dimension. Since the size of the space dimension is $N \times N$, DSW requires $N \times N \times N$ computations to compute each hypervolume. As explained in Section 2.2.2, to find the optimal warp path, the total computations required is $(2N \times b_x \times b_y)$. Therefore, the total computations of DSW is

$$2N^3 + (2N \times b_x \times b_y). \tag{6.1}$$

For DSW-based video detection application, the length of each subvideo of the target video is the same as the length of the query video.

6.2.2 Multiscale DTSW

For Multiscale-DTSW-based video detection application, we perform Multiscale DTSW algorithm for comparing each subvideo of the target video to the query video. Because we allow the time axis to warp, we do not have to set the length of each target subvideo to be the same as the length of the query video. Assume that the length of each target subvideo is p.

Figure 6-2 shows three possible categories of the optimal warp path in the time

dimension of comparing the target subvideo to the query video. If the target subvideo is too long to match the query video, then more than one frame at the end of the target subvideo will be forced to match with the last frame of the query video as shown in Figure 6-2(a). If the target subvideo is too short to match the query video, then more than one frame at the end of the query video will be forced to match with the last frame of the target subvideo as shown in Figure 6-2(c).

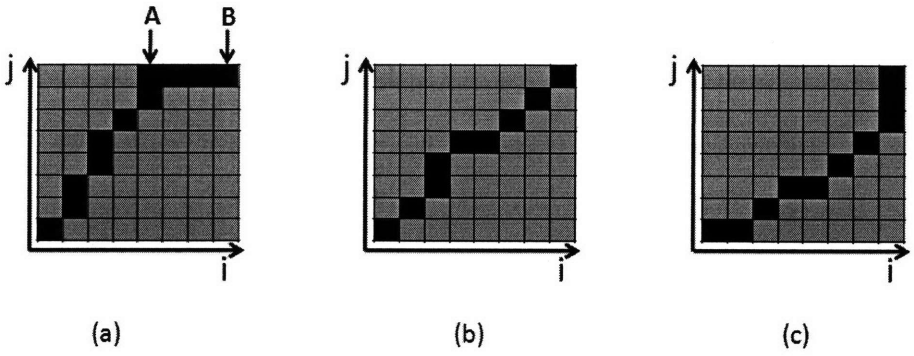

Figure 6-2: Three categories of the optimal warp path in the time dimension of matching a target subvideo to the query video. (a) The target subvideo is too long to match with the query video (b) The target subvideo matches the query video (c) The target subvideo is too short to match with the query video.

In Multiscale-DTSW-based video detection application, we set p with a big value such that the target subvideo will never be too short to match with the query video. Therefore, we only have a problem if the target subvideo is too long to match with the query video.

In Multiscale DTSW algorithm, we usually take the minimum distance along the top right corner ($i = I$ and $j = J$) of the Cum hypervolume as the distance between the two input videos, as shown as Point B in Figure 6-2(a). But in the video detection application, we should choose the minimum distance of the Cum hypervolume along Point A in Figure 6-2(a). The additional distance after Point A is due to the constraint of Multiscale DTSW that must return the optimal warp path that ends at $i = I$ and $j = J$. We do not want to include the additional distance to our video detection analysis. Therefore, we need to be able to find Point A correctly at the Cum hypervolume.

To solve this problem, we first find cells in the time dimension of the optimal warp path that are located at $j = J$. We then divide each $\{x, y\}$ plane along those cells with the length of the optimal warp path from cell {1,1} in time dimension to those cells. In Figure 6-2(a), we will divide the plane along Point A by 8, the plane along the cell next to Point A by 9, the plane along the cell before Point B by 10, and the plane along Point B by 11. The minimum $\{x, y\}$ plane among these planes will be the plane to be stacked in the *Detection Volume*.

6.2.3 Combination of DSW and Multiscale DTSW

For the third method, we combine DSW and Multiscale DTSW methods. We first find the temporal location of the query video inside the target video by using DSW-based video detection application. We then refine the solution by performing Multiscale-DTSW-based video detection application at the neighbors of the query temporal location found by DSW-based application.

The advantage of this method compare to Multiscale-DTSW-based application is its speed. The execution time of DSW-based application is much faster than Multiscale-DTSW-based application. The experimental results are provided in the next section. And the execution time of this third method is slower than DSW-based application due to the additional step of refining the solution, but it is faster than Multiscale-DTSW method. And because we allow temporal warping in the solution refinement step, the solution of the third method may be more accurate than the solution of DSW-based video detection application.

6.3 The Performance of the Video Detection Application

6.3.1 DSW-based Video Detection Application

Table 6.1 and Table 6.2 show the accuracy of the DSW-based video detection application on 40 target cases with various temporal and spatial offsets. The location of a

query video inside a target video is represented by two locations: location in the time dimension (temporal location) and location in the space dimension (spatial location). The detected or actual query temporal location is the starting frame of the subvideo of the target video that is similar to the query video. The detected or actual query spatial location is the x and y coordinates in the detected or actual temporal location frame of the target video where the first frame of the query video matches the similar-sized region in that frame. The x and y coordinates are the center coordinate of the region in the target frame that matches the query frame. The temporal detection offset means how many frames away from the true query temporal location that a detected query location is considered as a correct detection. The spatial detection offset means that how many pixels away from the true query spatial location that a detection query location is considered as a correct detection. The results suggest

Table 6.1: The accuracy in the time dimension of the DSW-based video detection application on 40 target cases with various temporal detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension).

Video set	Temporal Detection Offset					
	0	± 1	± 2	± 3	± 4	$\pm \geq 5$
walk	0.875	1	1	1	1	1
skip	0.5	1	1	1	1	1
side	0.625	0.875	1	1	1	1
run	0.25	0.75	1	1	1	1
jump	0.125	0.75	1	1	1	1
Average	0.475	0.875	1	1	1	1

that DSW-based video detection application is very good in detecting the temporal location of the query video inside the target video but the application is not as good in detecting the spatial location of the query video. The results show that the application is able to detect the downsampled query video accurately within 2 frames away from the true query temporal location. But within 2 pixels away from the true query spatial location, the application can only detect 22.5% of downsampled query videos correctly.

Although the application is not able to detect the spatial location accurately, but

Table 6.2: The accuracy in the space dimension of the DSW-based video detection application on 40 target cases with various spatial detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was 21 × 13 × 13 (x × y × time dimension).

Video set	Spatial Detection Offset					
	0	±1	±2	±3	±4	± ≥ 5
walk	0	0	0.125	0.5	1	1
skip	0.125	0.125	0.375	1	1	1
side	0	0	0	0	0.25	1
run	0	0	0.125	0.375	0.75	1
jump	0.125	0.125	0.5	1	1	1
Average	0.05	0.05	0.225	0.575	0.8	1

this implementation is fast in execution. The average execution time for one target case is only 33.4 seconds (running on a Pentium M laptop with 1.3 GHz processor and 768MB memory).

6.3.2 Multiscale-DTSW-based Video Detection Application

Table 6.3 and Table 6.4 show the accuracy of the Multiscale-DTSW-based video detection application on 40 target cases with various temporal and spatial offsets. The 40 target cases used in this experiment are the same target cases used in the experiments for DSW-based video detection application in the previous subsection.

Table 6.3: The accuracy in the time dimension of the Multiscale-DTSW-based video detection application on 40 target cases with various temporal detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was 21 × 13 × 13 (x × y × time dimension).

Video set	Temporal Detection Offset					
	0	±1	±2	±3	±4	± ≥ 5
walk	0.5	1	1	1	1	1
skip	0.25	0.875	1	1	1	1
side	0.125	0.75	1	1	1	1
run	0.125	0.75	1	1	1	1
jump	0.25	0.625	1	1	1	1
Average	0.25	0.8	1	1	1	1

The results suggest that Multiscale-DTSW-based video detection application is

Table 6.4: The accuracy in the space dimension of the Multiscale-DTSW-based video detection application on 40 target cases with various spatial detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension).

Video set	Spatial Detection Offset					
	0	± 1	± 2	± 3	± 4	$\pm \geq 5$
walk	0	0.5	1	1	1	1
skip	0.125	0.5	0.875	0.875	1	1
side	0	0	0	0	0	1
run	0	0	0.25	0.75	1	1
jump	0	0.25	0.375	0.375	0.75	1
Average	0.025	0.25	0.5	0.6	0.75	1

good in detecting the temporal location of the query video inside the target video. The results show that the application is able to detect the downsampled query video accurately within 2 frames away from the true query temporal location. The application performs better than DSW-based video detection application in detecting the spatial location of the query video. Within 2 pixels away from the true query spatial location, the application can detect 50% of downsampled query videos correctly.

Although this application is better than DSW-based application in detecting the spatial location of the query video, but this implementation is slow in execution. The average execution time for one target case is 1353.8 seconds (running on a Pentium M laptop with 1.3 GHz processor and 768MB memory). The slow execution time of Multiscale DTSW method compared to DSW method makes the Multiscale DTSW method less desirable for the video detection application.

6.3.3 DSW-and-Multiscale-DTSW-based Video Detection Application

Table 6.5 and Table 6.6 show the accuracy of the DSW-and-Multiscale-DTSW-based video detection application on 200 target cases with various temporal and spatial offsets. The average execution time for one target case is 216.65 seconds (running on a Pentium M laptop with 1.3 GHz processor and 768MB memory). We want to compare the performance of DSW-and-Multiscale-DTSW-based video detection ap-

plication with other methods. As explained in the previous subsection, the execution of Multiscale-DTSW-based application is slow, therefore we want to compare only the performance of DSW-based and DSW-and-Multiscale-DTSW-based applications. Table 6.7 and Table 6.8 show the accuracy of the DSW-based video detection application on the same 200 target cases used in the experiments for DSW-and-Multiscale-DTSW-based video detection application.

Table 6.5: The accuracy in the time dimension of the DSW-and-Multiscale-DTSW-based video detection application on 200 target cases with various temporal detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension).

Video set	Temporal Detection Offset					
	0	± 1	± 2	± 3	± 4	$\pm \geq 5$
walk	0.375	0.975	1	1	1	1
skip	0.4	0.95	1	1	1	1
side	0.275	0.85	1	1	1	1
run	0.275	0.75	0.975	1	1	1
jump	0.2	0.675	1	1	1	1
Average	0.305	0.84	0.995	1	1	1

Table 6.6: The accuracy in the space dimension of the DSW-and-Multiscale-DTSW-based video detection application on 200 target cases with various spatial detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension).

Video set	Spatial Detection Offset					
	0	± 1	± 2	± 3	± 4	$\pm \geq 5$
walk	0	0.475	0.975	1	1	1
skip	0	0.225	0.425	0.5	0.75	1
side	0	0	0.075	0.175	0.425	1
run	0	0.35	0.575	0.8	0.925	1
jump	0	0.55	0.575	0.75	0.875	1
Average	0	0.32	0.525	0.645	0.795	1

If we allow 2 frames tolerance in the detected query temporal location and 2 pixels tolerance in the detected query spatial location, both DSW-based and DSW-and-Multiscale-DTSW-based applications can detect the temporal location of the downsampled query video with 99.5% accuracy. However, for the spatial location, DSW-based video detection application can only detect 28.5% of the downsampled

Table 6.7: The accuracy in the time dimension of the DSW-based video detection application on 200 target cases with various temporal detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension).

Video set	Temporal Detection Offset					
	0	± 1	± 2	± 3	± 4	$\pm \geq 5$
walk	0.65	1	1	1	1	1
skip	0.425	0.925	0.975	1	1	1
side	0.55	0.9	1	1	1	1
run	0.175	0.6	0.875	1	1	1
jump	0.3	0.55	0.925	0.975	1	1
Average	0.42	0.795	0.955	0.995	1	1

Table 6.8: The accuracy in the space dimension of the DSW-based video detection application on 200 target cases with various spatial detection offsets. The optical flow component used was **u**. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension).

Video set	Spatial Detection Offset					
	0	± 1	± 2	± 3	± 4	$\pm \geq 5$
walk	0.025	0.125	0.35	0.6	1	1
skip	0	0.025	0.125	0.325	0.775	1
side	0	0	0	0.225	0.65	1
run	0	0	0.5	0.25	0.725	1
jump	0	0.2	0.45	0.725	0.9	1
Average	0.005	0.07	0.285	0.425	0.81	1

query videos while DSW-and-Multiscale-DTSW-based application can detect 52.5% of the downsampled query videos. The performance of DSW-and-Multiscale-DTSW is also better if we allow 3 frames and 3 pixels tolerance in the detected query temporal and spatial locations.

However, for tolerance of 4 frames and 4 pixels in the detected query temporal and spatial locations, the performance of DSW-based video detection application is better. It can detect the temporal location with 100% accuracy and the spatial location with 81% accuracy. The DSW-and-Multiscale-DTSW-based application can detect the temporal location with 100% accuracy and the spatial location with 79.5% accuracy.

In summary, for small temporal and spatial offset, we should use DSW-and-

Multiscale-DTSW method since the accuracy of the detection result is better even though the execution time is slower than the DSW method. However, for big temporal and spatial offsets, we should use DSW method because it is faster and more accurate.

6.3.4 Sensitivity to Scale Variance

We carried out experiments to investigate the sensitivity of our video detection application to scale variation. To obtain different scales for the query video, we downsampled or upsampled the query video in the space dimension and kept the target video unchanged.

Table 6.9 and Table 6.10 show the result of the video detection application with different scales of the query video.

Table 6.9: The average accuracy in the time dimension of the video detection application on 40 target cases for various scale differences between the query and target videos. The optical flow component used was $\mathbf{u}$. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). The method used was the combination of DSW and Multiscale DTSW.

Size of query	Percentage size of query	Temporal Detection Offset					
		0	± 1	± 2	± 3	± 4	$\pm \geq 5$
15×9	71.43%	0.375	0.825	1	1	1	1
17×11	80.95%	0.525	0.75	0.975	1	1	1
21×13	100%	0.475	0.875	1	1	1	1
29×17	138.10%	0.35	0.75	0.95	1	1	1

Table 6.10: The average accuracy in the space dimension of the video detection application on 40 target cases for various scale differences between the query and target videos. The optical flow component used was $\mathbf{u}$. Variable $b_x = 5$ and $b_y = 3$. Resolution was $21 \times 13 \times 13$ (x $\times$ y $\times$ time dimension). The method used was the combination of DSW and Multiscale DTSW.

Size of query	Percentage size of query	Spatial Detection Offset					
		0	± 1	± 2	± 3	± 4	$\pm \geq 5$
15×9	71.43%	0	0.1	0.175	0.25	0.35	1
17×11	80.95%	0.075	0.2	0.225	0.35	0.45	1
21×13	100%	0	0.275	0.5	0.575	0.75	1
29×17	138.10%	0	0.1	0.125	0.325	0.525	1

If we tolerate 5% error in the video detection result, then the video detection application's ability in detecting the temporal location of the query video is insensitive to a variation of 20% in the scale difference for temporal detection offset of more than 2 frames. The video detection application's ability in detecting the spatial location of the query video is sensitive to scale variance.

Chapter 7

Summary

Dynamic Time and Space Warping (DTSW) [1] is a technique used in video matching applications to find the optimal alignment between two videos. Because DTSW requires $O(N^4)$ time and space complexity, it is only suitable for short and coarse resolution videos. In this thesis, we introduce Multiscale DTSW: a modification of DTSW that has linear time and space complexity ($O(N)$) with good accuracy.

7.1 Contributions

Below is the list of the contributions of this research:

- We have introduced and implemented a new approach of comparing two videos, namely Multiscale DTSW. It is based on DTSW but has much less computational cost without much degradation in the performance. We have also shown the efficiency of Multiscale DTSW both theoretically and empirically.

- We have also introduced and implemented several extensions to Multiscale DTSW that can further increase its efficiency, namely Multiscale DTSW with Eigenframes implementation (Multiscale DTSWEF), Multiscale DTSW with Control Points, and Multiscale DTSW with Level Jump.

- We have implemented two applications (video classification application and video detection application) to show the efficiency of Multiscale DTSW. For

video detection application, we introduce a novel approach of combining DSW and Multiscale DTSW to find good solution in a fast manner.

- We have introduced a new way of comparing two videos by segmenting the videos into several parts, comparing each part individually, warping each part independently, and then combining all the parts back. By using this method, the two videos will be warped in a way that the warped videos are more similar to each other.

7.2 Future work

We did not try all possible combinations of the decision variables' values in the video classification application in deciding the best values for the decision variables. We chose the values of the decision variables in the order of motion scalar combination, resolution, b_x and b_y, and variance. However, this may not be the best way to determine the values of the variables. Therefore, the work can be extended to consider the video classification application as a system with numerous variables and certain range for each variable and to conduct the minimum number of experiments that will optimally choose the best value for each variable.

We have only implemented two applications that base on Multiscale DTSW as the video comparison algorithm. There are many applications that can be built on top of the Multiscale DTSW algorithm. For example, a video query application that takes a sample video clip as an input.

In the experiments for tolerance to scale variance of the video classification and detection applications, we have limited number of scale variances in the query video because we must downsample or upsample the query video with an integer value. To get more scale variances in the query video, we can resize the query video using an image processing software.

Because DTSW has a parallel structure that allows it to be implemented in a parallel manner, Multiscale DTSW can also be implemented in a parallel manner that will further explore its efficiency.

Appendix A

Target and query videos

Below are some target and query videos used in this research to evaluate the performance of Multiscale DTSW. The videos are represented by several key frames taken from the videos.

A.1 Karate punch videos

Figure A-1: Karate punch query video

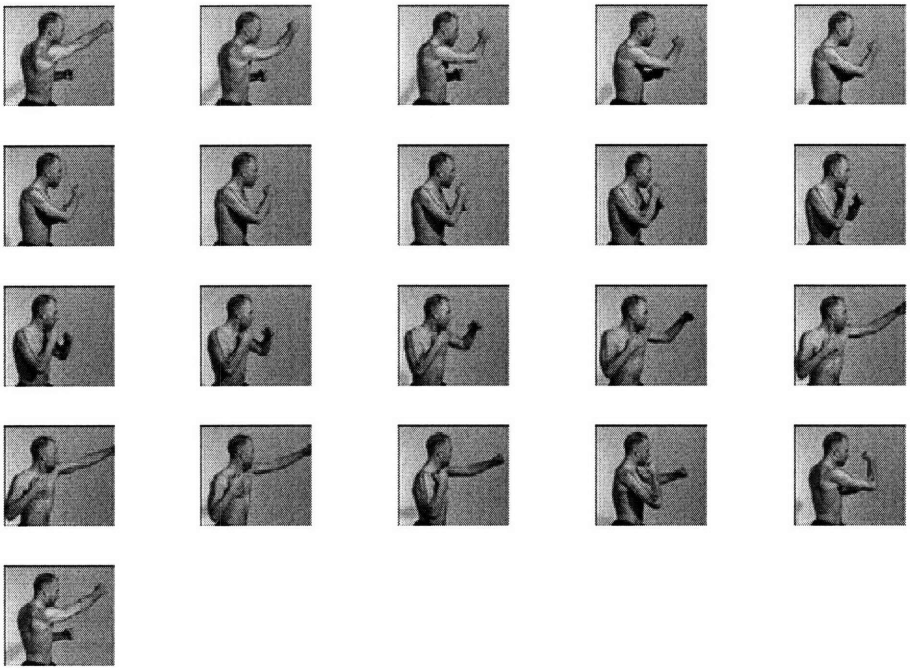

Figure A-2: Karate punch target video

A.2 Heart valve videos

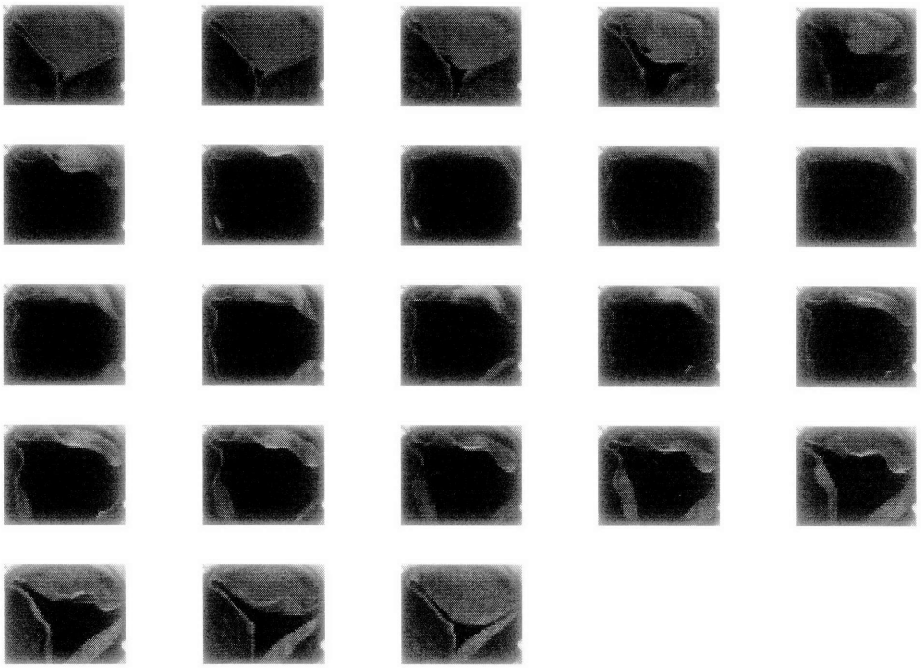

Figure A-3: Heart valve query video

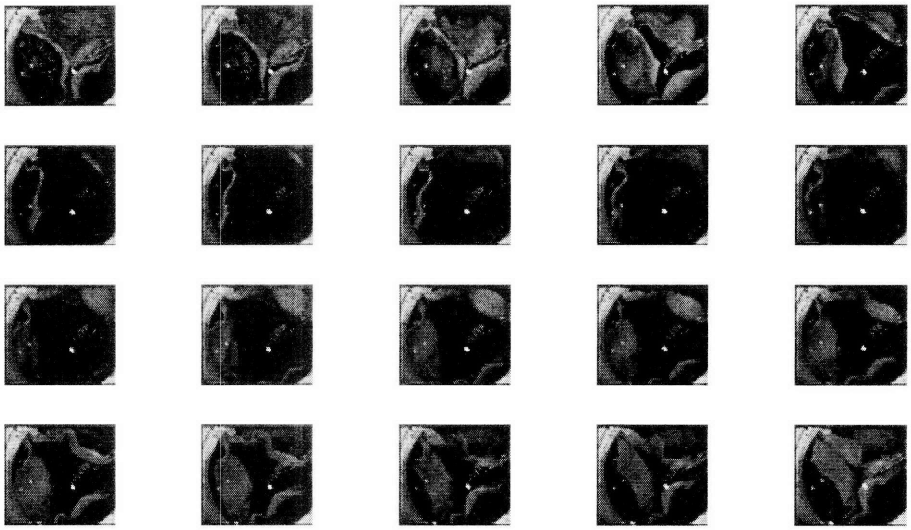

Figure A-4: Heart valve target video

A.3 Horse racing videos

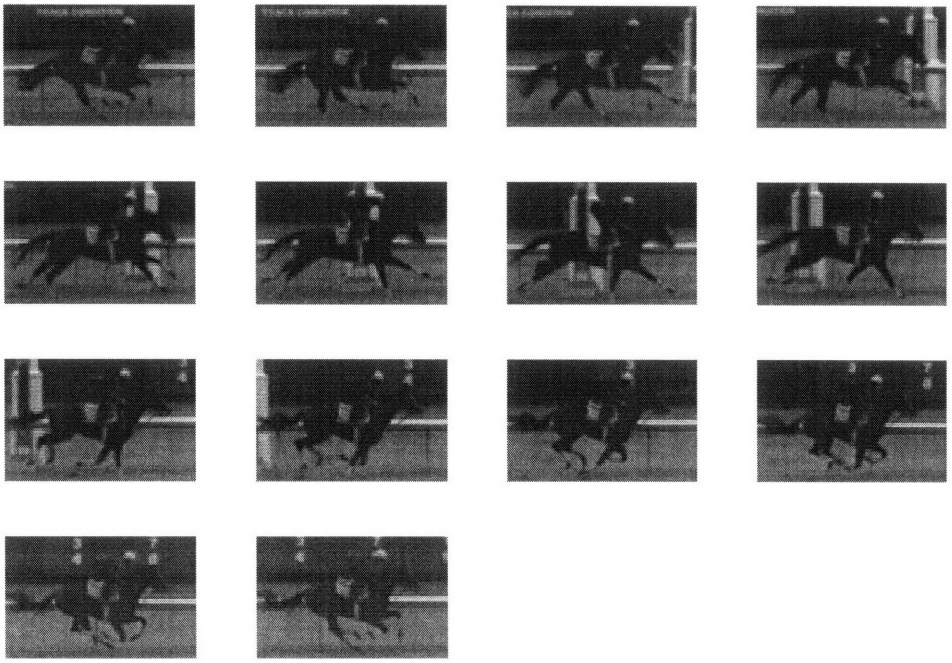

Figure A-5: Horse racing query video

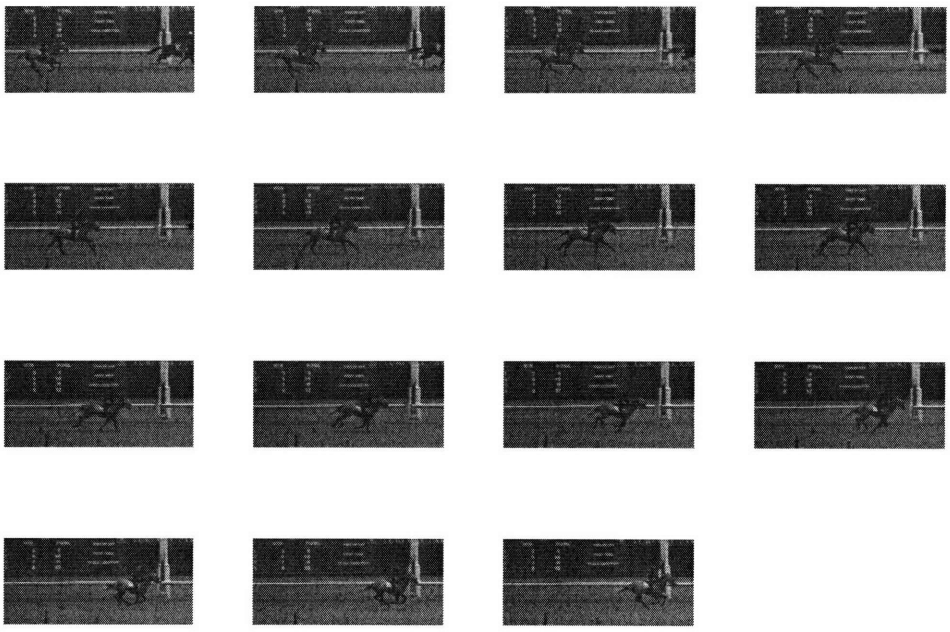

Figure A-6: Horse racing target video

A.4 Person walking videos

The following videos are taken from
www.wisdom.weizmann.ac.il/~vision/BehaviorCorrelation.html.

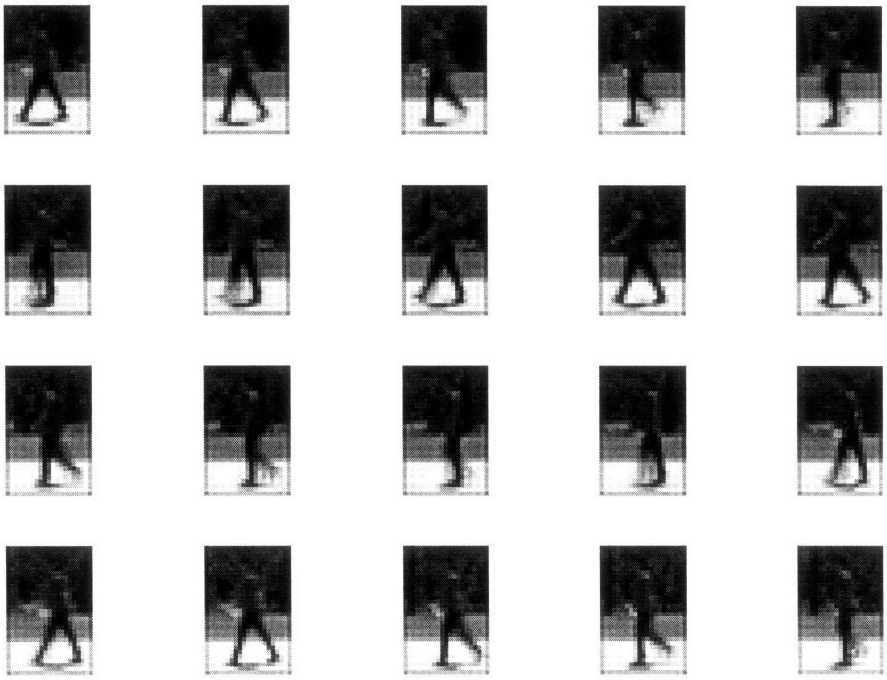

Figure A-7: A query video showing a person who is walking

Figure A-8: A target video showing a person walking on the beach

A.5 Palm opening and closing videos

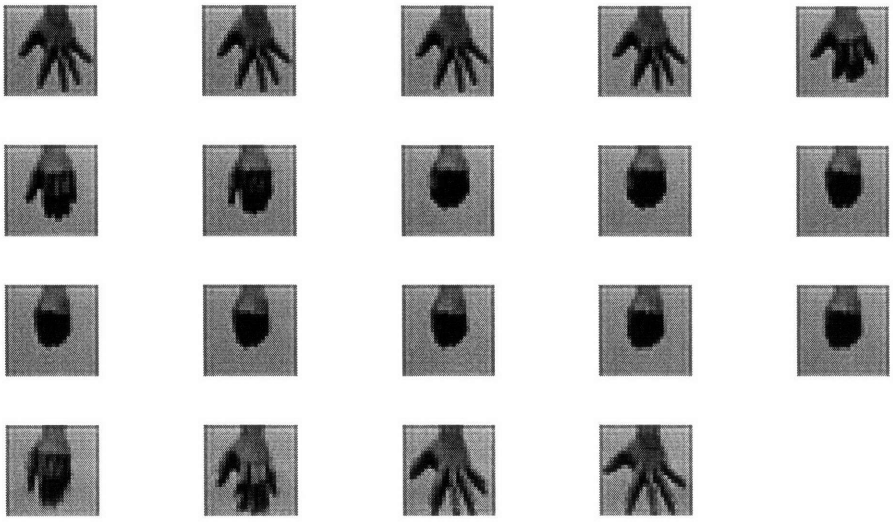

Figure A-9: Palm opening and closing query video

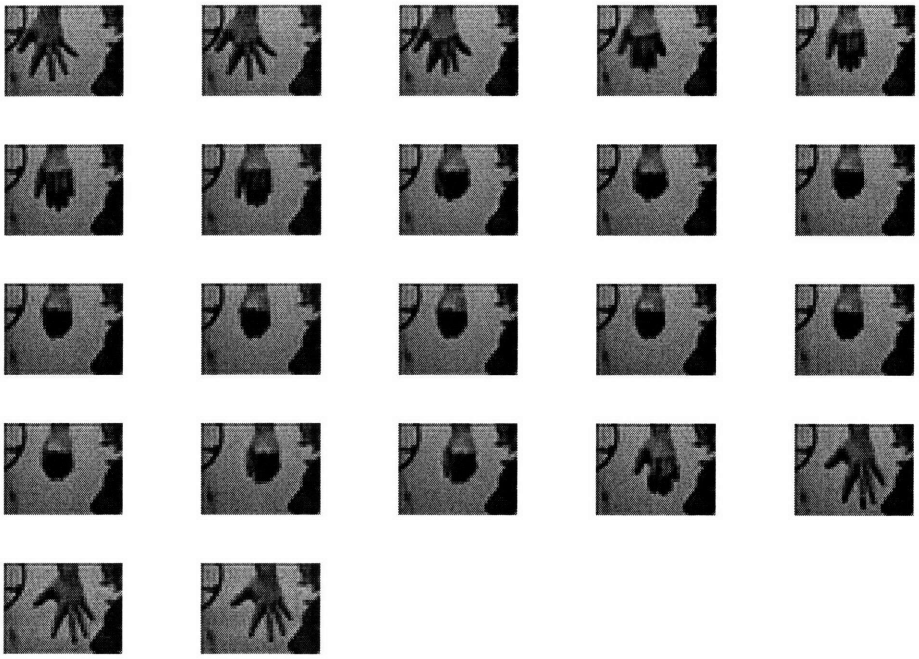

Figure A-10: Palm opening and closing target video

A.6 Random videos

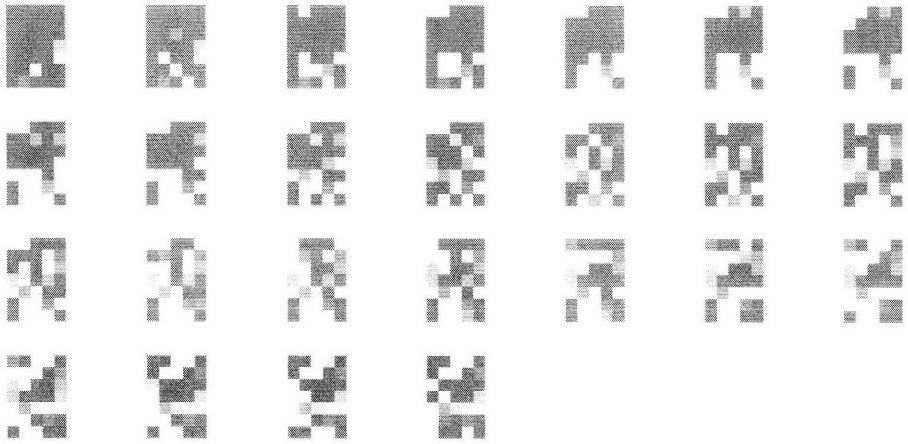

Figure A-11: Random query video

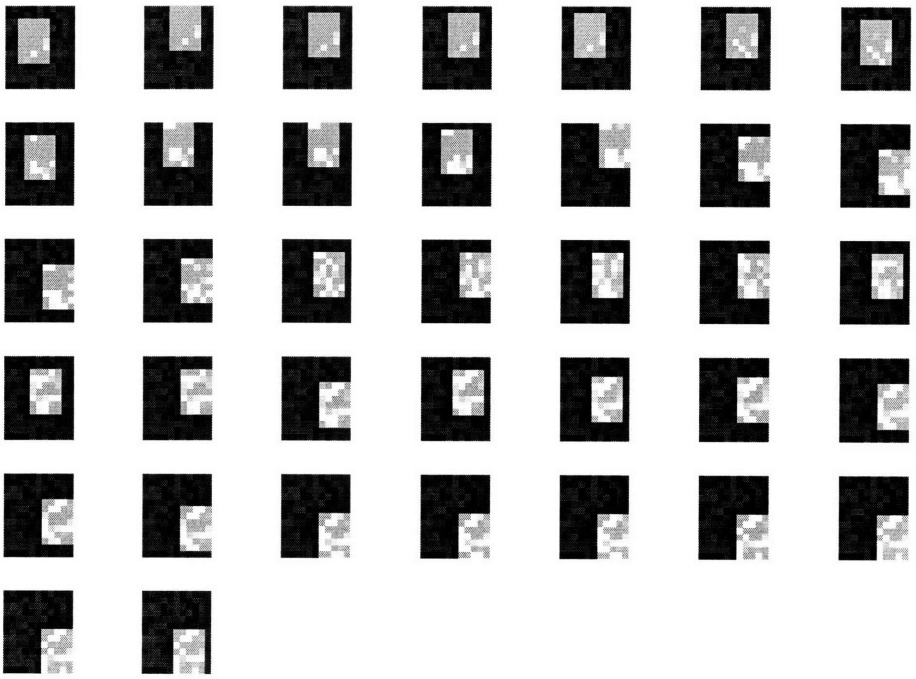

Figure A-12: Random target video

A.7 Video classification template videos

The following videos are taken from www.wisdom.weizmann.ac.il/~vision/BehaviorCorrelation.html and they are part of the template videos in the video classification application database to classify between walk, run, side, skip, and jump action. These videos are also used in the video detection application.

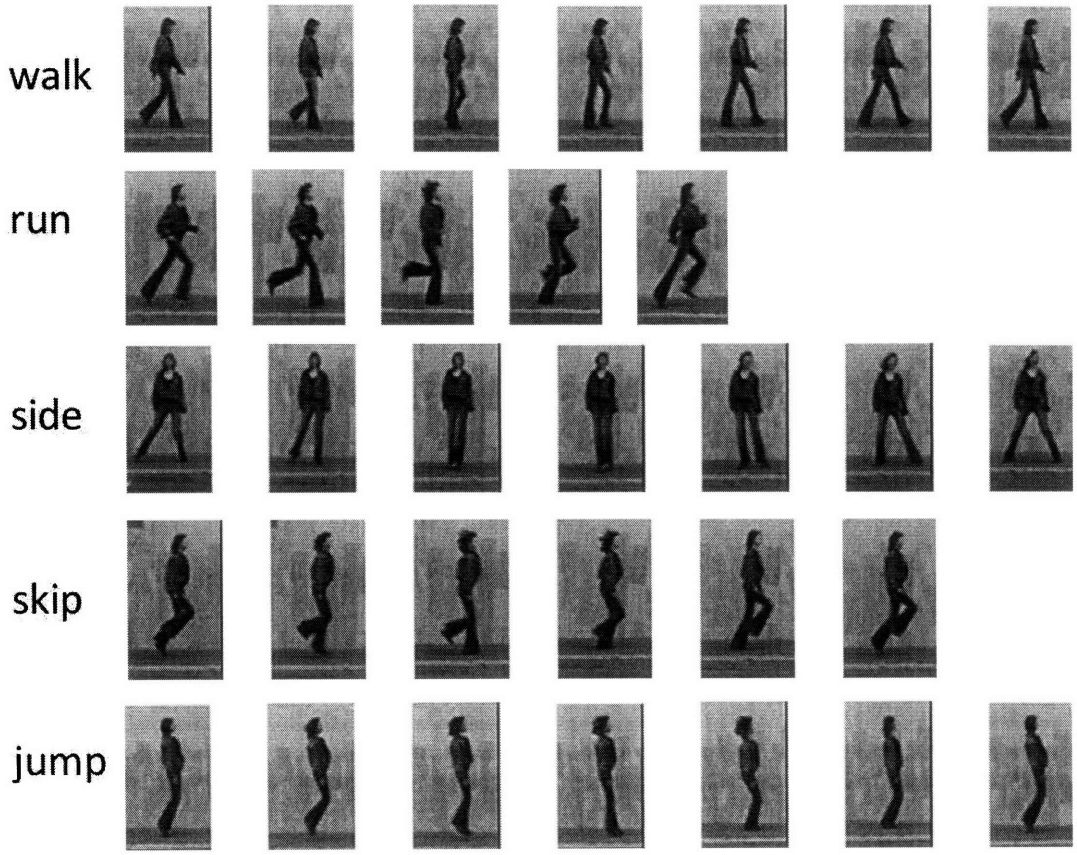

Figure A-13: A template video for each class in the video classification application database: walk, run, side, skip, and jump

The following videos are part of the template videos in the video classification application database to classify between left-to-right, circular, and hop pointing action.

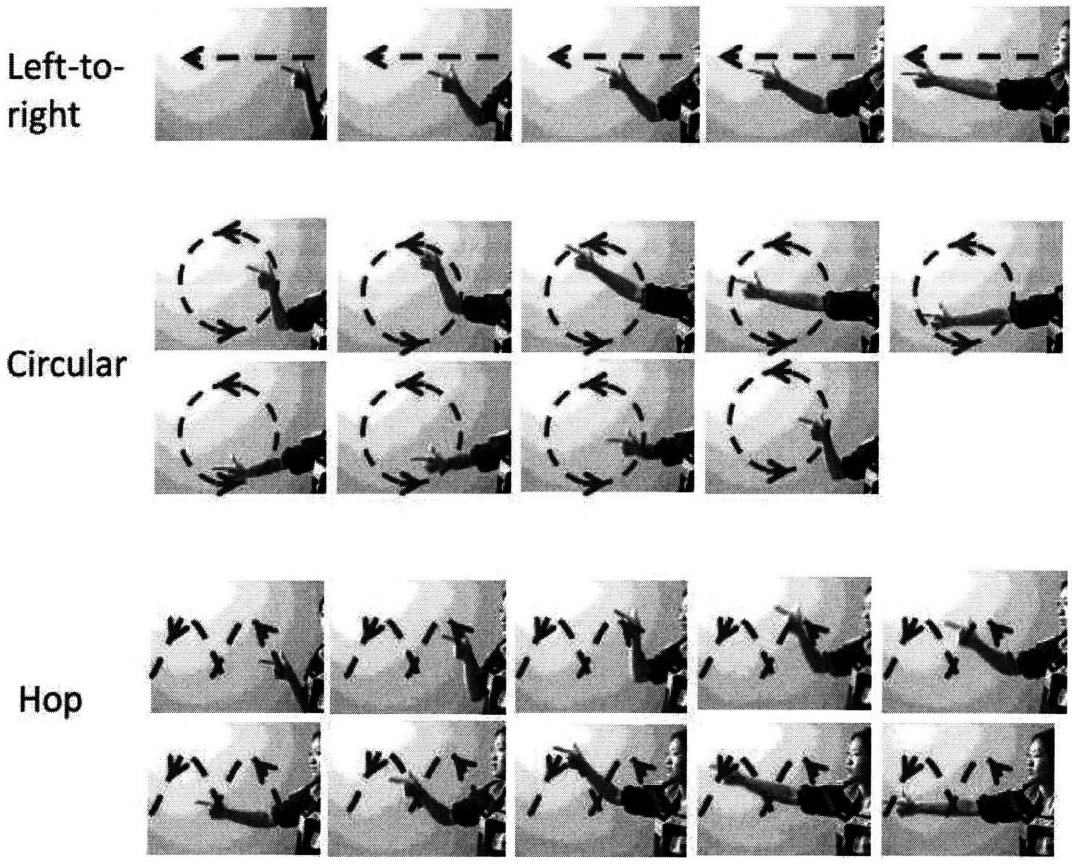

Figure A-14: A template video for each class in the video classification application database: left-to-right, circular, and hop

Appendix B

Piece-wise Multiscale DTSW on Karate Punch Videos

Piece-wise Multiscale DTSW applies Multiscale DTSW algorithm on each component of a target and query video and find the optimal warp path for each component. Piece-wise Multiscale DTSW then warps each component of the target video according to its warp path and combines all the warped components of the target video into a single video.

The similarity between the target video warped piece-wisely and the query video is then compared with the similarity between the target video warped wholly and the query video.

In our experiments, we divided the karate punch videos (target and query videos) into three parts: the head, the body and the arm of the karate instructor. For the first experiment, we searched for the optimal warp path for the full target video using Multiscale DTSW and then warped the target video according to the found optimal warp path. For the second experiment, we searched for the optimal warp path for each component of the target video using Multiscale DTSW and then warped each component according to its optimal warp path and then combined them into a single video.

Both warped target videos from the two experiments were then compared with the query video. The similarity between two videos is measured by the absolute

difference between the two videos. The absolute difference between two videos is defined as the summation of the absolute difference in the grayscale value of each pixel in each frame of the two videos. The higher the absolute difference is, the more dissimilar the two videos are. The absolute differences between the warped target video and the query video as well as the execution time for the two experiments are shown in Table B.1. From the table, we can observe that the warped target video found using Piece-wise Multiscale DTSW is more similar to the query video or Piece-wise Multiscale DTSW's solution is more accurate. However, the execution time of Piece-wise Multiscale DTSW is longer than of Multiscale DTSW.

Figure B-1 shows the query video. Figure B-2 shows the warped target video using Multiscale DTSW and Figure B-3 shows the warped target video using Piece-wise Multiscale DTSW.

Table B.1: Comparison of the similarity between the query video and the warped target video found using Multiscale DTSW and Piece-wise Multiscale DTSW. The similarity between two videos is measured by the absolute difference between the two videos. The absolute difference between two videos is defined as the summation of the absolute difference in the grayscale value of each pixel in each frame of the two videos. The higher the absolute difference is, the more dissimilar the two videos are. The execution time for each experiment running on an IBM Pentium M laptop (1.3 GHz processor with 768MB memory) is also included.

Method	Absolute difference	Execution time in seconds
Dimension = $11 \times 12 \times 12 \times 7$		
Multiscale DTSW	17714104	25.89
Piece-wise Multiscale DTSW	16251504	259.58
Dimension = $21 \times 23 \times 12 \times 7$		
Multiscale DTSW	30882268	64.00
Piece-wise Multiscale DTSW	27313873	1498.17

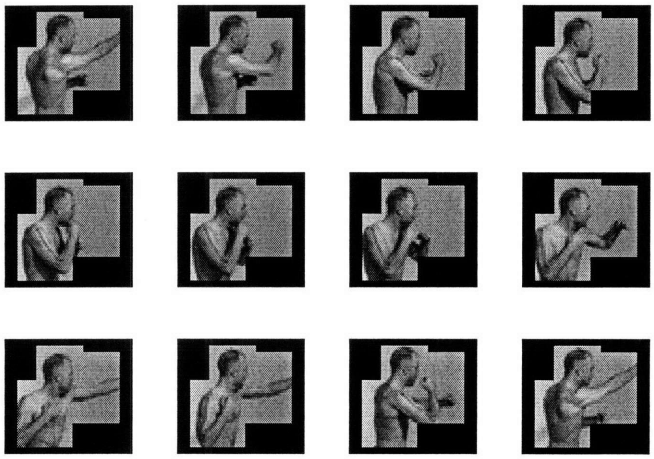

Figure B-1: Karate punch query video with the *Cum* hypervolume's dimension = $11 \times 12 \times 12 \times 7$

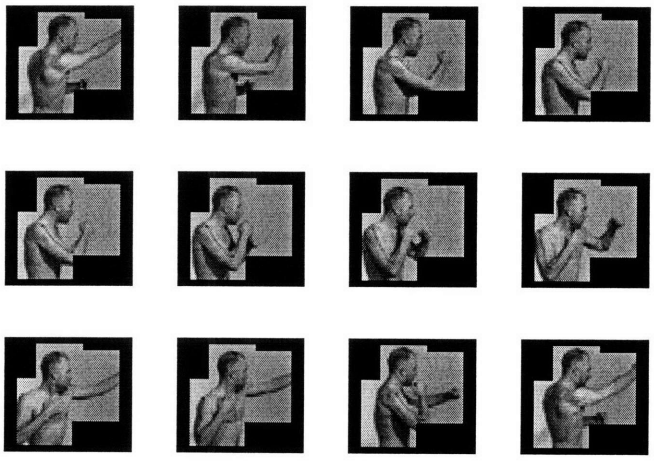

Figure B-2: Warped target video using Multiscale DTSW with the *Cum* hypervolume's dimension = $11 \times 12 \times 12 \times 7$

Figure B-3: Warped target video using Piece-wise Multiscale DTSW with the *Cum* hypervolume's dimension = $11 \times 12 \times 12 \times 7$

Bibliography

[1] Brian W. Anthony. *Video Based System Monitoring.* PhD thesis, Massachusetts Institute of Technology, 2006.

[2] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAI-94 Workshop on Knowledge Discovery in Databases (KDD-94)*, Seattle, 1994.

[3] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. In *International Conference on Computer Vision*, number 2, pages 1395–1402, 2005.

[4] S. Chu, E. Keogh, D. Hart, and Michael Pazzani. Iterative deepening dynamic time warping for time series. In *Second SIAM International Conference on Data Mining*, Arlington, Virginia, 2002.

[5] E. G. Ciani, A. Porta, G. Baselli, M. Turiel, S. Muzzupappa, F. Pieruzzi, C. Crema, A. Malliani, and S. Cerutti. Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. In *IEEE Computers in Cardiology*, number 25, NY, 1998.

[6] D. M. Gavrila and L. S. Davis. Towards 3-D model-based tracking and recognition of human movement: a multi-view approach. In *International Workshop on Automatic Face- and Gesture-Recognition*, Zurich, 1995.

[7] K. Gollmer and C. Posten. Detection of distorted pattern using dynamic time warping algorithm and application for supervision of bioprocesses. In *On-Line Fault Detection and Supervision in the Chemical Process Industries 4*, 1995.

[8] Hans D. Hohne, Cecil Coker, Stephen E. Levinson, and Lawrence R. Rabiner. On temporal alignment of sentences of natural and synthetic speech. In *IEEE Trans. Acoustics, Speech, and Signal Processing*, 1983.

[9] F. Itakura. Minimum prediction residual principle applied to speech recognition. In *IEEE Trans. Acoustics, Speech, and Signal Processing*, number 23.

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In *34th Design and Automation Conference*, pages 526–529, Anaheim, California, 1997.

[11] A. Kassidas, J. F. MacGregor, and P. A. Taylor. Synchronization of batch trajectories using dynamic time warping. In *American Institute of Chemical Engineers*, number 44, pages 864–887, 1998.

[12] E. Keogh, T. Palpanas, V. Zordan, and D. Gunopulos. Indexing large human-motion databases. In *International Conference on Very Large Databases (VLDB)*, 2004.

[13] E. Keogh and M. Pazzani. Scaling up dynamic time warping for datamining applications. In *6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, 2000.

[14] E. Keogh and M. Pazzani. Derivative dynamic time warping. In *SIAM International Conference on Data Mining*, 2001.

[15] Z. M. Kovacs-Vajna. A fingerprint verification system based on triangular matching and dynamic time warping. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, number 22, pages 1266–1276, November 2000.

[16] M. E. Munich and P.Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *8th IEEE International Conference on Computer Vision*, 1999.

[17] Cory Myers, Lawrence R. Rabiner, and Aaron E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. In *IEEE Trans. Acoustics, Speech, and Signal Processing*, 1980.

[18] T. Oates, M. D. Schmill, and P. R. Cohen. A method for clustering the experiences of a mobile robot that accords with human judgments. In *17th National Conference on Artificial Intelligence*, pages 846–851, 2000.

[19] L. Rabiner and B. Juang. *Fundamentals of speech recognition.* Prentice Hall, Englewood Cliffs, N.J, 1993.

[20] Michalis Raptis, Matteo Bustreo, and Stefano Soatto. Time warping under dynamic constraints. In *IEEE International Conference on Computer Vision*, Rio de Janeiro, Brazil, 2007.

[21] C. A. Ratanamahatana and E. Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data, in conjuction with the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2004)*, 2004.

[22] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recongition. In *IEEE Trans. Acoustics, Speech, and Signal Processing*, number 26, 1978.

[23] S. Salvador and P. Chan. FastDTW: Toward accurate dynamic time warping in linear time and space. In *KDD Workshop on Mining Temporal and Sequential Data*, pages 70–80, Seattle, 2004.

[24] M. Schmill, T. Oates, and P. Cohen. Learned models for continuous planning. In *Seventh International Workshop on Artificial Intelligence and Statistics*, 1999.

[25] Jonathan Shlens. A tutorial on principal component analysis. 2005.

[26] Lindsay I Smith. A tutorial on principal components analysis. February 2002.

[27] H. J. L. M. Vullings, M. H. G. Verhaegen, and H. B. Verbruggen. Ecg segmentation using time warping. In *Advances in Intelligent Data Analysis*, pages 275–285, 1997.

[28] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *ACM International Conference on Machine Learning*, pages 1033–1040, 2006.

[29] B. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *International Conference of Data Engineering*, 1998.