Beyond Dynamic Textures: a Family of Stochastic Dynamical Models
for Video with Applications to Computer Vision

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in
Electrical Engineering (Signal and Image Processing)

by

Antoni Bert Chan

Committee in charge:

Professor Nuno Vasconcelos, Chair
Professor Serge J. Belongie
Professor Kenneth Kreutz-Delgado
Professor Bhaskar D. Rao
Professor Lawrence K. Saul

2008
The dissertation of Antoni Bert Chan is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2008
TABLE OF CONTENTS

Signature Page ................................................................. iii
Table of Contents ................................................................... iv
List of Figures ........................................................................ ix
List of Tables ........................................................................... xii
Acknowledgements .................................................................... xiii
Vita and Publications ................................................................ xvii
Abstract of the dissertation ......................................................... xix

Chapter I  Introduction ......................................................... 1
  I.A.  Modeling motion ......................................................... 2
  I.B.  Contributions of the thesis ........................................... 4
    I.B.1.  Probabilistic kernels for dynamic textures ............... 4
    I.B.2.  Multi-modal extensions of the dynamic texture .......... 5
    I.B.3.  Non-linear extension of the dynamic texture .......... 6
    I.B.4.  Applications to computer vision and computer audition . 7
  I.C.  Organization of the thesis ............................................. 8

Chapter II  Modeling motion flow with dynamic textures ............... 10
  II.A.  Modeling motion flow ............................................... 11
  II.B.  Dynamic textures ..................................................... 12
    II.B.1.  Probability distributions .................................... 14
  II.C.  Inference ................................................................. 16
    II.C.1.  Kalman filter .................................................... 16
    II.C.2.  Kalman smoothing filter .................................... 17
  II.D.  Parameter estimation ............................................... 18
    II.D.1.  Expectation maximization .................................. 18
    II.D.2.  Least-squares .................................................. 19
    II.D.3.  On-line estimation with least-squares .................... 22
  II.E.  Acknowledgements .................................................. 26

Chapter III  Probabilistic kernels for the classification of dynamic textures 27
  III.A.  Introduction .......................................................... 28
  III.B.  Support vector machines and probabilistic kernels ......... 30
  III.C.  Kullback-Leibler divergence .................................... 31
    III.C.1.  Projection between state spaces ......................... 32
    III.C.2.  KL divergence between state spaces ..................... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V.C.2</td>
<td>E-Step</td>
<td>114</td>
</tr>
<tr>
<td>V.C.3</td>
<td>M-Step</td>
<td>115</td>
</tr>
<tr>
<td>V.C.4</td>
<td>Initialization strategies</td>
<td>115</td>
</tr>
<tr>
<td>V.D.</td>
<td>Approximate inference by Gibbs sampling</td>
<td>117</td>
</tr>
<tr>
<td>V.D.1</td>
<td>Sampling from $p(Z</td>
<td>X,Y)$</td>
</tr>
<tr>
<td>V.D.2</td>
<td>Sampling from $p(X</td>
<td>Z,Y)$</td>
</tr>
<tr>
<td>V.E.</td>
<td>Inference by variational approximation</td>
<td>119</td>
</tr>
<tr>
<td>V.E.1</td>
<td>Approximate factorial posterior distribution</td>
<td>120</td>
</tr>
<tr>
<td>V.E.2</td>
<td>Approximate inference</td>
<td>122</td>
</tr>
<tr>
<td>V.F.</td>
<td>Temporally-switching layered dynamic textures</td>
<td>124</td>
</tr>
<tr>
<td>V.F.1</td>
<td>Joint distribution of the TS-LDT</td>
<td>125</td>
</tr>
<tr>
<td>V.F.2</td>
<td>Parameter estimation with the EM algorithm</td>
<td>126</td>
</tr>
<tr>
<td>V.F.3</td>
<td>Inference by variational approximation</td>
<td>127</td>
</tr>
<tr>
<td>V.G.</td>
<td>Experimental evaluation of approximate inference and EM</td>
<td>129</td>
</tr>
<tr>
<td>V.G.1</td>
<td>Synthetic dataset</td>
<td>129</td>
</tr>
<tr>
<td>V.G.2</td>
<td>Inference experiments</td>
<td>130</td>
</tr>
<tr>
<td>V.G.3</td>
<td>EM experiments</td>
<td>131</td>
</tr>
<tr>
<td>V.H.</td>
<td>Application to motion segmentation</td>
<td>132</td>
</tr>
<tr>
<td>V.H.1</td>
<td>Results on synthetic circular motion</td>
<td>133</td>
</tr>
<tr>
<td>V.H.2</td>
<td>Results on synthetic texture database</td>
<td>135</td>
</tr>
<tr>
<td>V.H.3</td>
<td>Results on real video</td>
<td>139</td>
</tr>
<tr>
<td>V.I.</td>
<td>Summary and discussion</td>
<td>141</td>
</tr>
<tr>
<td>V.J.</td>
<td>Acknowledgements</td>
<td>142</td>
</tr>
<tr>
<td>V.K.</td>
<td>Appendix</td>
<td>143</td>
</tr>
<tr>
<td>V.K.1</td>
<td>Derivations for the LDT</td>
<td>143</td>
</tr>
<tr>
<td>V.K.2</td>
<td>Derivations for the TS-LDT</td>
<td>150</td>
</tr>
<tr>
<td>Chapter VI</td>
<td>Kernel dynamic textures</td>
<td>158</td>
</tr>
<tr>
<td>VI.A.</td>
<td>Introduction</td>
<td>159</td>
</tr>
<tr>
<td>VI.B.</td>
<td>Kernel PCA</td>
<td>160</td>
</tr>
<tr>
<td>VI.B.1</td>
<td>Learning KPCA coefficients</td>
<td>161</td>
</tr>
<tr>
<td>VI.B.2</td>
<td>Reconstruction from KPCA coefficients</td>
<td>161</td>
</tr>
<tr>
<td>VI.C.</td>
<td>Kernel dynamic textures</td>
<td>163</td>
</tr>
<tr>
<td>VI.C.1</td>
<td>Synthetic examples</td>
<td>163</td>
</tr>
<tr>
<td>VI.C.2</td>
<td>Tangent distance kernels</td>
<td>166</td>
</tr>
<tr>
<td>VI.C.3</td>
<td>Related works</td>
<td>167</td>
</tr>
<tr>
<td>VI.D.</td>
<td>The Martin distance between kernel dynamic textures</td>
<td>168</td>
</tr>
<tr>
<td>VI.E.</td>
<td>Applications to video classification</td>
<td>170</td>
</tr>
<tr>
<td>VI.E.1</td>
<td>Panning video texture databases</td>
<td>170</td>
</tr>
<tr>
<td>VI.E.2</td>
<td>Experimental setup</td>
<td>171</td>
</tr>
<tr>
<td>VI.E.3</td>
<td>Classification results</td>
<td>172</td>
</tr>
<tr>
<td>VI.F.</td>
<td>Summary and discussion</td>
<td>175</td>
</tr>
<tr>
<td>VI.G.</td>
<td>Acknowledgements</td>
<td>176</td>
</tr>
</tbody>
</table>
Chapter IX  Application: Modeling music with dynamic textures . . . . . . 246
  IX.A.  Introduction .................................................... 247
  IX.B.  Dynamic texture models for music .......................... 248
    IX.B.1.  Song segmentation ......................................... 249
  IX.C.  Experimental evaluation ........................................ 249
    IX.C.1.  Song database .............................................. 250
    IX.C.2.  Audio features .............................................. 250
    IX.C.3.  Song segmentation ........................................... 251
    IX.C.4.  Boundary detection ......................................... 252
    IX.C.5.  Song segment retrieval .................................... 254
  IX.D.  Summary and discussion ...................................... 255
  IX.E.  Acknowledgements ............................................. 256

Chapter X  Conclusions ................................................. 257

Bibliography .......................................................... 259
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.1</td>
<td>Example of visual processes</td>
<td>2</td>
</tr>
<tr>
<td>II.1</td>
<td>Graphical model for the dynamic texture</td>
<td>13</td>
</tr>
<tr>
<td>II.2</td>
<td>Example of the dynamic texture model</td>
<td>14</td>
</tr>
<tr>
<td>III.1</td>
<td>Example of the KL divergence rate</td>
<td>39</td>
</tr>
<tr>
<td>III.2</td>
<td>Frames from a video of highway traffic</td>
<td>42</td>
</tr>
<tr>
<td>III.3</td>
<td>Examples from the databases used for evaluation</td>
<td>43</td>
</tr>
<tr>
<td>III.4</td>
<td>Precision-recall curve for traffic video retrieval</td>
<td>45</td>
</tr>
<tr>
<td>III.5</td>
<td>Retrieval results on traffic video</td>
<td>46</td>
</tr>
<tr>
<td>III.6</td>
<td>Evaluation of the KL kernel on two databases</td>
<td>47</td>
</tr>
<tr>
<td>III.7</td>
<td>Classification of traffic congestion</td>
<td>48</td>
</tr>
<tr>
<td>III.8</td>
<td>Classification of traffic congestion spanning 15 hours</td>
<td>49</td>
</tr>
<tr>
<td>IV.1</td>
<td>Graphical model for a mixture of dynamic textures</td>
<td>66</td>
</tr>
<tr>
<td>IV.2</td>
<td>Time-series clustering results on three synthetic problems</td>
<td>81</td>
</tr>
<tr>
<td>IV.3</td>
<td>Clustering of traffic video</td>
<td>83</td>
</tr>
<tr>
<td>IV.4</td>
<td>Segmentation of synthetic video</td>
<td>86</td>
</tr>
<tr>
<td>IV.5</td>
<td>Segmentation of ocean-fire</td>
<td>87</td>
</tr>
<tr>
<td>IV.6</td>
<td>Results on the synthetic database</td>
<td>89</td>
</tr>
<tr>
<td>IV.7</td>
<td>Segmentation examples from the synthetic texture database</td>
<td>91</td>
</tr>
<tr>
<td>IV.8</td>
<td>Segmentation examples from the synthetic texture database</td>
<td>92</td>
</tr>
<tr>
<td>IV.9</td>
<td>Segmentation of a fountain scene</td>
<td>93</td>
</tr>
<tr>
<td>IV.10</td>
<td>Segmentation of traffic scenes</td>
<td>94</td>
</tr>
<tr>
<td>IV.11</td>
<td>Segmentation of two pedestrian scenes</td>
<td>94</td>
</tr>
<tr>
<td>V.1</td>
<td>Generative model for a video with multiple dynamic textures</td>
<td>108</td>
</tr>
<tr>
<td>V.2</td>
<td>Graphical model for the layered dynamic texture</td>
<td>109</td>
</tr>
<tr>
<td>V.3</td>
<td>Graphical model for the variational approximation of the LDT</td>
<td>120</td>
</tr>
<tr>
<td>V.4</td>
<td>Tradeoff between runtime and segmentation performance using approximate inference</td>
<td>131</td>
</tr>
<tr>
<td>V.5</td>
<td>Segmentation of synthetic circular motion</td>
<td>133</td>
</tr>
<tr>
<td>V.6</td>
<td>LDT results on synthetic texture database</td>
<td>137</td>
</tr>
<tr>
<td>V.7</td>
<td>Results on the synthetic texture database</td>
<td>138</td>
</tr>
<tr>
<td>V.8</td>
<td>Segmentation of ocean-fire with LDT</td>
<td>138</td>
</tr>
<tr>
<td>V.9</td>
<td>Segmentation of a ferris wheel and windmill</td>
<td>139</td>
</tr>
<tr>
<td>V.10</td>
<td>Segmentation of a whirlpool using LDT</td>
<td>140</td>
</tr>
<tr>
<td>V.11</td>
<td>Examples of motion segmentation using LDT</td>
<td>141</td>
</tr>
<tr>
<td>VI.1</td>
<td>Examples of synthesis with the kernel dynamic texture</td>
<td>165</td>
</tr>
<tr>
<td>VI.2</td>
<td>Examples from the UCLA-pan-1 video texture database</td>
<td>170</td>
</tr>
</tbody>
</table>
Figure VI.3  Classification results on panning video textures........173
Figure VI.4  Misclassification of chaotic water..................175

Figure VII.1  A scene with dynamic background....................184
Figure VII.2  Overview of generalized Stauffer-Grimson background modeling for dynamic textures.................190
Figure VII.3  ROC curves for foreground detection on Bottle video.197
Figure VII.4  Foreground detection results on Bottle sequence.198
Figure VII.5  ROC curves for foreground detection on Boats........199
Figure VII.6  ROC curves for foreground detection with DTM and approximate DTM..................200

Figure VII.7  Foreground detection results on Boats1.............201
Figure VII.8  Foreground detection results on Boats2.............202
Figure VII.9  Foreground detection results on PETS2005...........204
Figure VII.10 Foreground detection results on a beach scene........204
Figure VII.11 Foreground detection results on Chopper video.......204
Figure VII.12 Foreground detection results on Skiing video.........206
Figure VII.13 Foreground detection results on CycleFire video....206

Figure VIII.1  Examples of a low-resolution scene containing a sizable crowd with inhomogeneous dynamics............213
Figure VIII.2  Correspondence between crowd size and two simple features........215
Figure VIII.3  Overview of the crowd counting system...............216
Figure VIII.4  Perspective map.......................................217
Figure VIII.5  Ground-truth annotations for pedestrian databases221
Figure VIII.6  Ground-truth pedestrian counts....................224
Figure VIII.7  Crowd motion segmentation.........................226
Figure VIII.8  Crowd counting results on Peds1....................229
Figure VIII.9  Cumulative error on Peds1............................230
Figure VIII.10 Error splits for different crowd sizes on Peds1......230
Figure VIII.11 Test error versus training size.....................230
Figure VIII.12 Crowd counting examples from Peds1................231
Figure VIII.13 Count estimates on 55 minutes of Peds1.............233
Figure VIII.14 Example counting results on 55 minutes of Peds1.....234
Figure VIII.15 Crowd counting results on Peds2........................237
Figure VIII.16 Cumulative error on Peds2............................238
Figure VIII.17 Error splits for different crowd sizes on Peds1......238
Figure VIII.18 Crowd counting examples from Peds2................239
Figure VIII.19 Count estimates on 55 minutes of Peds2.............240
Figure VIII.20 Example counting results on 55 minutes of Peds2.....241
Figure VIII.21 People detection results on Peds1 and Peds2...........242
Figure VIII.22 Counting results on Peds1 using people detection....244
Figure VIII.23 Counting results on Peds2 using people detection....244
Figure VIII.24 Error breakdown for people detection..................244
Figure IX.1  Example song segmentation using DTM . . . . . . . . . . . . . . . . . . 253
Figure IX.2  2-D visualization of the distribution of song segments . . 255
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table III.1</td>
<td>Confusion matrix for traffic congestion classification</td>
<td>48</td>
</tr>
<tr>
<td>Table IV.1</td>
<td>Notation for EM for DTM</td>
<td>68</td>
</tr>
<tr>
<td>Table IV.2</td>
<td>Overall Rand index on the three synthetic experiments</td>
<td>82</td>
</tr>
<tr>
<td>Table IV.3</td>
<td>Videos in motion segmentation experiment</td>
<td>85</td>
</tr>
<tr>
<td>Table IV.4</td>
<td>Average Rand index on the synthetic texture database</td>
<td>88</td>
</tr>
<tr>
<td>Table V.1</td>
<td>Comparison of approximate inference methods on synthetic data</td>
<td>130</td>
</tr>
<tr>
<td>Table V.2</td>
<td>Average Rand index on the synthetic texture database</td>
<td>135</td>
</tr>
<tr>
<td>Table VI.1</td>
<td>Classification results on panning video databases</td>
<td>174</td>
</tr>
<tr>
<td>Table VI.2</td>
<td>Modifications for kernel centering</td>
<td>180</td>
</tr>
<tr>
<td>Table VII.1</td>
<td>Quantitative results on three water scenes</td>
<td>199</td>
</tr>
<tr>
<td>Table VIII.1</td>
<td>Properties of the Peds1 database</td>
<td>222</td>
</tr>
<tr>
<td>Table VIII.2</td>
<td>Properties of the Peds2 database</td>
<td>222</td>
</tr>
<tr>
<td>Table VIII.3</td>
<td>Comparison of feature sets and regression methods on Peds1</td>
<td>227</td>
</tr>
<tr>
<td>Table VIII.4</td>
<td>Comparison of feature sets on Peds2</td>
<td>235</td>
</tr>
<tr>
<td>Table VIII.5</td>
<td>Counting results using people detection</td>
<td>243</td>
</tr>
<tr>
<td>Table IX.1</td>
<td>Song segmentation using MFCC features</td>
<td>252</td>
</tr>
<tr>
<td>Table IX.2</td>
<td>Song segmentation using Chroma features</td>
<td>252</td>
</tr>
<tr>
<td>Table IX.3</td>
<td>Boundary detection using MFCC features</td>
<td>254</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to acknowledge many people for helping me during my doctoral work. Their help and support was instrumental in the completion of this work.

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Nuno Vasconcelos, for his guidance, inspiration, and accessibility throughout my graduate career. His exceptional insight into the fundamentals of computer vision and machine learning are, without a doubt, reflected in this thesis. I am also grateful for having an exceptional doctoral committee, and wish to thank Professor Serge J. Belongie, Professor Kenneth Kreutz-Delgado, Professor Bhaskar D. Rao, and Professor Lawrence K. Saul for their valuable input, lively discussions, and accessibility.

My research was supported in part by National Science Foundation (NSF) award IIS-0534985 (PI: Nuno Vasconcelos) and NSF Integrative Graduate and Research Traineeship (NSF-IGERT) award DGE-0333451 (PIs: Virginia de Sa, Garrison Cottrell). I would like to thank the PIs and NSF for providing me with the opportunity to work in this area.

I would like to thank all my colleagues at SVCL, Dr. Dashan Gao, Dr. Gustavo Carneiro, Sunhyoung Han, Vijay Mahadevan, Hamed Masnadi-Shirazi, and Nikhil Rasiwasia, for their support, friendship, and assistant throughout the years. I would also like to thank Zhang-Sheng John Liang, Vijay Mahadevan, Luke Barrington, Dr. Piotr Dollar, and Professor Gert R. G. Lanckriet for their collaboration on some of the experiments, and Mulloy Morrow and Jeffrey Cuenco for their assistance in obtaining video and ground-truth data. I also thank Dr. Pedro J. Moreno, Professor Stefano Soatto, and Professor Rene Vidal for their insightful discussions.

Over the past years, I have made many friends in San Diego: Luke Barrington, Doug and Megan Turnbull, Carolina Galleguillos, Dennis Franco, Dr. Patrick Amihood, Dr. Dave Wipf, Dr. Qinghua Zhao, Dr. Laia Egea-Pujol, Chloe
Chang, Moira MacKinnon, friends of the Audition Laboratory, SO3, the OV house (past and present), and many others. I thank them all for their support and distractions.

I would like to thank my parents for their love and support throughout my educational career, without which none of this would be possible. Last, and certainly not least, I owe many thanks to my wife¹, Janet Hui-wen Hsiao, for her love, support, inspiration, motivation, and cooking. 我愛你

The text of Section II.D.3 is based on the material as it appears in: A. B. Chan, V. Mahadevan, and N. Vasconcelos, “Generalized Stauffer-Grimson background subtraction for dynamic scenes”, submitted for publication to Machine Vision and Applications. The dissertation author was a primary researcher and an author of the cited material.


¹as of May 3, 2009.
material.


The text of Chapter VI, in part, is based on the material as it appears in: A. B. Chan and N. Vasconcelos, “Classifying video with kernel dynamic textures”, in *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), June 2007. The dissertation author was a primary researcher and an author of the cited material.

The text of Chapter VII, in full, is based on the material as it appears in: A. B. Chan, V. Mahadevan, and N. Vasconcelos, “Generalized Stauffer-Grimson background subtraction for dynamic scenes”, submitted for publication to *Machine Vision and Applications*. The dissertation author was a primary researcher and an author of the cited material.

The text of Chapter VIII, in part, is based on the material as it appears in: A. B. Chan, Z. S. J. Liang, and N. Vasconcelos. “Privacy preserving crowd monitoring: counting people without people models or tracking”, in *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), June 2008. The dissertation author was a primary researcher and an author of the cited material.

The text of Chapter IX, in full, is based on the material as it appears in: A. B. Chan, L. Barrington, and G. R. G. Lanckriet, “Dynamic texture models of music”, submitted for publication to *International Conference on Acoustics, Speech, and Signal Processing* (ICASSP), 2009. The dissertation author was a
primary researcher and joint first author of the cited material.
VITA

2000  Bachelor of Science, *Magna Cum Laude*  
Electrical Engineering, Cornell University, Ithaca, NY

2001  Master of Engineering  
Electrical Engineering, Cornell University, Ithaca, NY

2001–2003  Visiting Scientist  
Vision and Image Analysis Group  
Department of Electrical and Computer Engineering  
Cornell University, Ithaca, NY

2003–2008  Research Assistant  
Statistical and Visual Computing Laboratory  
Department of Electrical and Computer Engineering  
University of California, San Diego

2008  Doctor of Philosophy  
Electrical and Computer Engineering, University of California, San Diego

PUBLICATIONS


ABSTRACT OF THE DISSERTATION

Beyond Dynamic Textures: a Family of Stochastic Dynamical Models for Video with Applications to Computer Vision

by

Antoni Bert Chan

Doctor of Philosophy in Electrical Engineering
(Signal and Image Processing)

University of California, San Diego, 2008
Professor Nuno Vasconcelos, Chair

One family of visual processes that has relevance for various applications of computer vision is that of, what could be loosely described as, visual processes composed of ensembles of particles subject to stochastic motion. The particles can be microscopic (e.g. plumes of smoke), macroscopic (e.g. leaves blowing in the wind), or even objects (e.g. a human crowd or a traffic jam). The applications range from remote monitoring for the prevention of natural disasters (e.g. forest fires), to background subtraction in challenging environments (e.g. outdoor scenes with moving trees in the background), and to surveillance (e.g. traffic monitoring). Despite their practical significance, the visual processes in this family still pose tremendous challenges for computer vision. In particular, the stochastic nature of the motion fields tends to be highly challenging for traditional motion representations such as optical flow, parametric motion models, and object tracking. Recent efforts have advanced towards modeling video motion probabilistically,
by viewing video sequences as “dynamic textures” or, more precisely, samples from a generative, stochastic, texture model defined over space and time. Despite its successes in applications such as video synthesis, motion segmentation, and video classification, the dynamic texture model has several major limitations, such as an inability to account for visual processes consisting of multiple co-occurring textures (e.g. smoke rising from a fire), and an inability to model complex motion (e.g. panning camera motion).

In this dissertation, we propose a family of dynamical models for video that address the limitations of the dynamic texture, and apply these new models to challenging computer vision problems. In particular, we introduce two multi-modal models for video, the mixture of dynamic textures and the layered dynamic texture, which provide principled frameworks for video clustering and motion segmentation. We also propose a non-linear model, the kernel dynamic texture, which can capture complex patterns of motion through a non-linear manifold embedding. We present a new framework for the classification of dynamic textures, which combines the modeling power of the dynamic texture and the generalization guarantees, for classification, of the support vector machine classifier, by deriving a new probabilistic kernel based on the Kullback-Leibler divergence between dynamic textures. Finally, we demonstrate the applicability of these models to a wide variety of real-world computer vision problems, including motion segmentation, video clustering, video texture classification, highway traffic monitoring, crowd counting, and adaptive background subtraction. We also demonstrate that the dynamic texture is a suitable representation for musical signals, by applying the proposed models to the computer audition task of song segmentation. These successes validate the dynamic texture framework as a principled approach for representing video, and suggest that the models could be useful in other domains, such as computer audition, that require the analysis of time-series data.
Chapter I

Introduction
I.A Modeling motion

One family of visual processes that has relevance for various applications of computer vision is that of, what could be loosely described as, visual processes composed of *ensembles of particles subject to stochastic motion*. Figure I.1 presents a few examples of these types of visual processes. The particles can be microscopic, e.g. plumes of smoke, macroscopic, e.g. leaves and vegetation blowing in the wind, or even objects, e.g. a human crowd, a flock of birds, a traffic jam, or a beehive. The applications range from remote monitoring for the prevention of natural disasters, e.g. forest fires, to background subtraction in challenging environments, e.g. outdoors scenes with vegetation, and various type of surveillance, e.g. traffic monitoring, homeland security applications, or scientific studies of animal behavior.

![Figure I.1 Examples of visual processes that are challenging for traditional spatio-temporal representations: fire, smoke, the flow of a river stream, or the motion of an ensemble of objects, e.g. a flock of birds, a bee colony, a school of fish, the traffic on a highway, or the flow of a crowd.](image)

Despite their practical significance, and the ease with which they are perceived by biological vision systems, the visual processes in this family still pose tremendous challenges for computer vision. In particular, the *stochastic nature* of the associated motion fields tends to be *highly challenging for traditional motion representations* such as optical flow \([1, 2, 3, 4]\), which requires some degree of motion smoothness, parametric motion models \([5, 6, 7]\), which assume a piece-wise planar world \([8]\), or object tracking \([9, 10, 11]\), which tends to be impractical when the number of subjects to track is large and these objects interact in a complex
manner.

The main limitation of all these representations is that they are inherently local, aiming to achieve understanding of the whole by modeling the motion of the individual particles. This is contrary to how these visual processes are perceived by biological vision: smoke is usually perceived as a whole, a tree is normally perceived as a single object, and the detection of traffic jams rarely requires tracking individual vehicles. Recently, there has been an effort to advance towards this type of holistic modeling, by viewing video sequences derived from these processes as dynamic textures or, more precisely, samples from stochastic processes defined over space and time [12, 13, 14, 15, 16, 17]. The dynamic texture (DT) [13] is an auto-regressive random process (specifically, a linear dynamical system) that contains two random variables: an observed variable, which determines the appearance of each video frame, and a hidden-state variable, which captures the dynamics (motion) of the video over time. Both the hidden state and the observed variable are representative of the entire image, enabling a holistic characterization of the motion for the entire sequence. The dynamic texture framework has been shown to have great potential for video synthesis [13], image registration [14], motion segmentation [15, 17, 18, 19], and video classification [16, 20]. This is, in significant part, due to the fact that the underlying generative probabilistic framework is capable of 1) abstracting a wide variety of complex motion patterns into a simple spatio-temporal process, and 2) synthesizing samples of the associated time-varying texture.

Despite these successes, the dynamic texture model has several major limitations. First, the distance functions proposed in the literature for classifying DT do not fully exploit the probabilistic framework of the dynamic texture. In particular, the existing distances are based on subspace angles [16] or trajectories [20], which do not take into account the stochastic nature of the texture. This is analogous to comparing Gaussian distributions using only the distance between their means. Clearly, the effectiveness of this distance will suffer when the vari-
ances of the Gaussians are discriminant. Second, the dynamic texture is unable to account for visual processes consisting of multiple, co-occurring, dynamic textures, for example, a flock of birds flying in front of a water fountain, highway traffic moving at different speeds, smoke rising from a fire, or video containing both trees in the background and people in the foreground. In such cases, the existing DT model is inherently ill-equipped to model the video, since it must represent multiple motion fields with a single dynamic process. Third, the dynamic texture can only model video where the motion is smooth, i.e. video textures where the pixel values change smoothly. This limitation stems from the linear assumptions of the model, specifically, the linear transitions between hidden-states and the linear subspace in which the video frames are represented. As a result, the dynamic texture is not capable of modeling more complex motion, such as chaotic motion (e.g. turbulent water) or camera motion (e.g. panning, zooming, and rotations).

I.B Contributions of the thesis

In this thesis we address the limitations of the dynamic texture through the development of new dynamical models for video. In contrast to previous work, our approach will focus on fully exploiting the dynamic texture as a generative probabilistic model. This will result in new robust models for video that provide a natural probabilistic framework for solving challenging computer vision problems. The main contributions of the thesis are as follows.

I.B.1 Probabilistic kernels for dynamic textures

We present a new framework for the classification of dynamic textures, which combines the modeling power of the dynamic texture and the generalization guarantees, for classification, of the support vector machine classifier. This combination is achieved by the derivation of a new probabilistic kernel based on the Kullback-Leibler divergence (KL) between dynamic textures. In particular,
we derive the KL-kernel for dynamic textures in both 1) the image space, which describes both the motion and appearance components of the spatio-temporal process; and 2) the hidden state space, which describes the temporal component alone. Together, the two kernels cover a large variety of video classification problems, including the cases where classes can differ in both appearance and motion, and the cases where appearance is similar for all classes and only motion is discriminant. We demonstrate the practical feasibility of the proposed classification framework for vision problems involving complex spatio-temporal visual stimuli, such as the classification of video textures, or the classification of patterns of highway traffic flow under variable environmental conditions. The new framework is shown to perform well above the state of the art and to produce quite promising results for difficult problems, such as monitoring highway congestion.

### I.B.2 Multi-modal extensions of the dynamic texture

To address the limitations of modeling multiple visual processes, we propose two novel multi-modal representations of video that extend the dynamic texture. The first proposed representation is the *mixture of dynamic textures*, which models an ensemble of video sequences as samples from a single probabilistic model, containing a finite collection of dynamic textures. We derive an expectation-maximization (EM) algorithm for learning the parameters of the model, and we relate the model to previous works in linear systems, machine learning, time-series clustering, control theory, and computer vision. Through extensive experimentation, we show that the mixture of dynamic textures is a suitable representation for both the appearance and dynamics of a variety of visual processes that have traditionally been challenging for computer vision (e.g. fire, steam, water, vehicle and pedestrian traffic, etc.). When compared with state-of-the-art methods in motion segmentation, including both temporal texture methods and traditional representations (e.g. optical flow or other localized motion representations), the mixture of dynamic textures achieves superior performance in the problems of clustering
and segmenting video of such processes.

For the second multi-modal representation, we propose to model a video containing co-occurring textures as a collection of stochastic layers of different appearance and dynamics, which we denote as the *layered dynamic texture* (LDT). Each layer is modeled as a temporal texture sampled from a different dynamic texture. The LDT model includes these systems, a collection of hidden layer assignment variables (which control the assignment of pixels to layers), and a Markov random field prior on these variables (which encourages smooth segmentations). We derive an EM algorithm for maximum-likelihood estimation of the model parameters from training video. Because exact inference is intractable, we also introduce two approximate inference procedures: a Gibbs sampler and a computationally efficient variational approximation. We experimentally study the trade-off between the quality of the two approximations and their complexity, and we evaluate the ability of the LDT to segment both synthetic and natural video into layers of coherent appearance and dynamics. These experiments show that the model possesses an ability to group regions of *globally homogeneous*, but *locally heterogeneous*, stochastic dynamics currently unparalleled in the literature.

### I.B.3 Non-linear extension of the dynamic texture

We address the smoothness limitations of the dynamic texture by improving the modeling capability of the dynamic texture, specifically, by adopting a non-linear observation function in the dynamic texture model. The non-linear function is represented with kernel principal component analysis (KPCA), and the resulting *kernel dynamic texture* (KDT) is capable of modeling a wider range of video motion, such as chaotic motion (e.g. turbulent water) or camera motion (e.g. panning). We build a video classifier based on the kernel dynamic texture, by deriving an appropriate distance function between KDT. Finally, we evaluate the efficacy of the model through a classification experiment on video containing camera motion.
I.B.4 Applications to computer vision and computer audition

We apply the proposed models to several challenging problems in computer vision and computer audition. In particular, we demonstrate state-of-the-art results in classification of video, including video textures and patterns of highway traffic flow under variable environmental conditions. We also demonstrate state-of-the-art performance in motion segmentation on a variety of visual processes, using the mixture of dynamic textures and the layered dynamic texture. Finally, we adopt the dynamic texture framework in three additional applications in computer vision and computer audition, which are described below.

We propose an adaptive model for backgrounds containing significant stochastic motion (e.g. water). The new model is based on a generalization of the Stauffer-Grimson background model, where each mixture component is modeled as a dynamic texture. We derive an on-line K-means algorithm for updating the parameters using a set of sufficient statistics of the model. Finally, we report on experimental results, which show that the proposed background model both quantitatively and qualitatively outperforms state-of-the-art methods in scenes containing significant background motions.

We present a vision system for estimating the size of inhomogeneous crowds, composed of pedestrians that travel in different directions, without using explicit object segmentation or tracking. First, the crowd is segmented into components of homogeneous motion, using the mixture of dynamic textures motion model. Second, a set of simple holistic features is extracted from each segmented region, and the correspondence between features and the number of people per segment is learned with Gaussian process regression. We validate both the crowd segmentation algorithm, and the crowd counting system, on two large pedestrian datasets. Finally, we present results of the system running on two full hours of video.

Finally, we consider representing a musical signal as a dynamic texture, a model for both the timbral and rhythmical qualities of sound. We apply the new
representation to the task of automatic song segmentation. In particular, we cluster sequences of audio feature-vectors, extracted from the song, using the mixture of dynamic textures. We show that the mixture model can both detect transition boundaries and accurately cluster coherent segments. The similarities between the dynamic textures which define these segments are based on both timbral and rhythmic qualities of the music, indicating that the model simultaneously captures two of the important aspects required for automatic music analysis.

I.C Organization of the thesis

The rest of the thesis is organized as follows. In Chapter II, we review the dynamic texture and the associated inference and learning algorithms. These algorithms are the basic building blocks for the algorithms proposed in the remaining chapters. In Chapter III, we introduce a probabilistic kernel function for the classification of dynamic textures. We derive the Kullback-Leibler divergence between dynamic textures, and perform experiments on video texture classification and traffic congestion analysis. In Chapter IV, we introduce the mixture of dynamic textures, which models a collection of video sequences as samples from a set of dynamic textures. We derive the associated EM learning algorithm, which provides a natural framework for clustering video sequences. The mixture model is applied to video clustering and motion segmentation. In Chapter V, we propose the layered dynamic texture, which represents video as a collection of stochastic layers of different appearance and dynamics. The EM learning algorithm is derived, along with two approximate inference algorithms, Gibbs sampling and a variational approximation. The two inference algorithms are compared experimentally, and the layered DT is applied to motion segmentation of video. In Chapter VI, we propose the kernel dynamic texture, which improves the modeling power of the standard dynamic texture by adopting a non-linear observation function. We derive a distance function between kernel dynamic textures, and apply the
model to classify video textures undergoing panning. In the remaining chapters, we apply these dynamic texture models to various problems in computer vision and computer audition. In Chapter VII, we propose an adaptive model for backgrounds containing significant stochastic motion (e.g. water), which is based on online estimation of dynamic textures. In Chapter VIII, we present a system for estimating the size of inhomogeneous crowds, composed of pedestrians that travel in different directions, without using explicit object segmentation or tracking. A key component of the system is the robust motion segmentation algorithm based on the mixture of dynamic textures. In Chapter IX, we represent a musical signal as a dynamic texture, and perform song segmentation with the mixture of dynamic textures. Finally, conclusions are provided in Chapter X.
Chapter II

Modeling motion flow with
dynamic textures
II.A Modeling motion flow

Figure I.1 presents a sample from a large collection of visual processes that have proven remarkably challenging for traditional motion representations, based on modeling of the individual trajectory of pixels [2, 3], particles [9], or objects in a scene. Since most of the information required for the perception of these processes is contained in the interaction between the many motions that compose them, they require a holistic representation of the associated motion field capable of capturing its variability without the need for segmentation or tracking of individual components. Throughout the years, some representations appeared particularly promising in this respect, e.g. the representation of the motion field as a collection of layers [6]. However, only recently some real success has been demonstrated through the modeling of these processes as dynamic textures, i.e. realizations of an auto-regressive stochastic process with both a spatial and temporal component [21, 13]. Like many other recent advances in vision, the success of these methods derives from the adoption of representations based on generative probabilistic models that can be learned from collections of training examples.

Various representations of a video sequence as a spatio-temporal texture have been proposed in the vision literature over the last decade. Earlier efforts were aimed at the extraction of features that capture both the spatial appearance of a texture and the associated motion flow field. For example, in [22], temporal textures are represented by the first and second order statistics of the normal flow of the video. These types of strictly feature-based representation can be useful for recognition but do not provide a probabilistic model that could be used for kernel design.

More recently, various authors proposed to model a temporal texture as a generative process, resulting in representations that can be used for both synthesis and recognition. One example is the multi-resolution analysis tree method of [23], which represents a temporal texture as the hierarchical multi-scale transform asso-
associated with a 3D wavelet. The conditional probability distributions of the wavelet coefficients in the tree are estimated from a collection of training examples and the texture is synthesized by sampling from this model. Another possibility is the spatio-temporal autoregressive (STAR) model of [21], which models the interaction of pixels within a local neighborhood over both space and time. By relying on spatio-temporally localized image features these representations are incapable of abstracting the video into a pair of holistic appearance and motion components.

This problem is addressed by the dynamic texture model of [13], an autoregressive random process (specifically, a linear dynamical system) that includes a hidden state variable, in addition to the observation variable that determines the appearance component. The motion flow of the video is captured by a dynamic generative model, from which the hidden state vector is drawn. The observation vector is then drawn from a second generative model, conditioned on the state variable. Both the hidden state vector and the observation vector are representative of the entire image, enabling a holistic characterization of the motion for the entire sequence. For this reason, we adopt the dynamic texture representation in the remainder of this work. In the remainder of this chapter, we discuss the dynamic texture model, along with algorithms for inference and learning. The dynamic texture, and the associated algorithms, will form the basic building blocks for methods proposed in later chapters.

II.B Dynamic textures

A dynamic texture [12, 13] (DT) is a generative model for both the appearance and the dynamics of video sequences. It consists of a random process containing an observed variable $y_t$, which encodes the appearance component (vectorized video frame at time $t$), and a hidden state variable $x_t$, which encodes the dynamics (evolution of the video over time). The state and observed variables are
related through the linear dynamical system (LDS) defined by

\begin{align*}
x_t &= Ax_{t-1} + v_t, \\
y_t &= Cx_t + w_t + \bar{y},
\end{align*}

where \( x_t \in \mathbb{R}^n \) and \( y_t \in \mathbb{R}^m \) (typically \( n \ll m \)). The parameter \( A \in \mathbb{R}^{n \times n} \) is a state transition matrix, \( C \in \mathbb{R}^{m \times n} \) is an observation matrix (e.g. containing the principal components of the video sequence when learned with [13]), and \( \bar{y} \) is the mean of the dynamic texture. The driving noise process \( v_t \) is normally distributed with zero mean and covariance \( Q \), i.e. \( v_t \sim \mathcal{N}(0, Q) \) where \( Q \in \mathbb{S}_+^n \) is a positive-definite \( n \times n \) matrix. The observation noise \( w_t \) is also zero mean and Gaussian, with covariance \( R \), i.e. \( w_t \sim \mathcal{N}(0, R) \) where \( R \in \mathbb{S}_+^m \) (typically \( R \) is assumed i.i.d., \( R = r I_m \)). In this thesis, we will specify the initial condition of the model by a probability distribution on \( x_1 \). In [13], the initial condition is specified by a fixed initial vector \( x_0 \in \mathbb{R}^n \), or equivalently \( x_1 \sim \mathcal{N}(\mu, Q) \), where \( \mu = Ax_0 \). In later chapters, we consider an extension of the original model [13] to accommodate an initial state \( x_1 \) of arbitrary mean and covariance, i.e. \( x_1 \sim \mathcal{N}(\mu, S) \). This extension produces a richer video model that can capture variability in the initial frame, which is necessary for learning a dynamic texture from multiple video samples (e.g. for the clustering and segmentation problems explored in Chapter IV). The dynamic texture is specified by the parameters \( \Theta = \{ A, Q, C, R, \mu, S, \bar{y} \} \), and can be represented by the graphical model of Figure II.1.

One interpretation of the DT model, when the columns of \( C \) are orthonormal (e.g. when learned with [13]), is that the video frame at time \( t \), \( y_t \), is a
Figure II.2 Example of the dynamic texture model: (top-left) frames from a traffic sequence; (bottom-left) the first three principal components; (right) the state space trajectory of the corresponding coefficients.

linear combination of the principal components stored in the columns of $C$. Under this interpretation, the state vector $x_t$ is the set of PCA coefficients of each video frame, and evolves according to a Gauss-Markov process (in time). Figure II.2 shows an example of a traffic sequence, its first three principal components (first three columns of $C$), and the corresponding state space coefficients.

An alternative interpretation considers a single pixel as it evolves over time. Each dimension of the state vector $x_t$ defines a one-dimensional temporal trajectory, and the pixel value is a weighted sum of these trajectories, according to the weighting coefficients in the corresponding row of $C$. This is analogous to the discrete Fourier transform, where a 1-D signal is represented as a weighted sum of complex exponentials but, for the DT, the trajectories are not necessarily orthogonal. This interpretation illustrates the ability of the DT to model a given motion at different intensity levels (e.g. cars moving from the shade into sunlight) by simply scaling rows of $C$.

II.B.1 Probability distributions

We next discuss the probability distributions of the dynamic texture. It can be shown [24, 25] that the distributions of the initial state, the conditional
state, and the conditional observation for the system defined in (II.1, II.2) are

\[
p(x_1) = G(x_1, \mu, S),
\]
\[
p(x_t|x_{t-1}) = G(x_t, Ax_{t-1}, Q),
\]
\[
p(y_t|x_t) = G(y_t, Cx_t + \bar{y}, R),
\]

where \( G(x, \mu, \Sigma) = \frac{1}{(2\pi)^{-n/2} |\Sigma|^{-1/2}} e^{-\frac{1}{2} \|x-\mu\|_\Sigma^2} \) is the \( n \)-dimensional multivariate Gaussian distribution, and \( \|x\|_\Sigma^2 = x^T \Sigma^{-1} x \) is the Mahalanobis distance with respect to the covariance matrix \( \Sigma \). Defining the mean and covariance of the state \( x_t \) and image \( y_t \) as

\[
\mu_t = \mathbb{E}[x_t], \quad S_t = \text{cov}(x_t), \quad \gamma_t = \mathbb{E}[y_t], \quad \Sigma_t = \text{cov}(y_t),
\]

it can also be shown [24] that the marginal distribution of state \( x_t \) is Gaussian with mean \( \mu_t \) and covariance \( S_t \), computed recursively according to

\[
\mu_t = A\mu_{t-1}, \quad S_t = AS_{t-1}A^T + Q,
\]

with initial conditions \( \mu_1 = \mathbb{E}[x_1] = \mu \) and \( S_1 = \text{cov}(x_1) = S \). The marginal distribution of \( y_t \) is also Gaussian, with mean \( \gamma_t \) and covariance \( \Sigma_t \) given by

\[
\gamma_t = C\mathbb{E}[x_t] = C\mu_t + \bar{y}, \quad \Sigma_t = C\text{cov}(x_t)C^T + R = CS_tC^T + R.
\]

Defining \( x_{1:\tau} = (x_1, \cdots, x_\tau) \) as a length-\( \tau \) sequence of state vectors, the distribution of the state sequence \( x_{1:\tau} \) is

\[
p(x_{1:\tau}) = p(x_1) \prod_{t=2}^{\tau} p(x_t|x_{t-1}).
\]

Defining the image sequence \( y_{1:\tau} = (y_1, \cdots, y_\tau) \), the joint distribution of states and images under the DT model is

\[
p(x_{1:\tau}, y_{1:\tau}) = p(x_1) \prod_{t=2}^{\tau} p(x_t|x_{t-1}) \prod_{t=1}^{\tau} p(y_t|x_t).
\]
II.C Inference

In this section, we briefly review the Kalman filter and Kalman smoothing filter for computing the probability distribution of the hidden state \(x_t\), conditioned on an observed sequence, \(y_{1:t}\). The Kalman filter can also be used to compute the log-likelihood, \(\log p(y_{1:t})\), of the observed sequence.

II.C.1 Kalman filter

The Kalman filter computes the mean and covariance of the state \(x_t\) of an LDS, conditioned on the current observed sequence \(y_{1:t} = (y_1, \ldots, y_t)\). We first define the following conditional expectations:

\[
\hat{x}_{t|t-1} = \mathbb{E}(x_t | y_{1:t-1}), \quad \text{(II.12)}
\]
\[
\hat{V}_{t|t-1} = \text{cov}(x_t | y_{1:t-1}) = \mathbb{E}((x_t - \hat{x}_{t|t-1})(x_t - \hat{x}_{t|t-1})^T | y_{1:t-1}), \quad \text{(II.13)}
\]
\[
\hat{x}_{t|t} = \mathbb{E}(x_t | y_{1:t}), \quad \text{(II.14)}
\]
\[
\hat{V}_{t|t} = \text{cov}(x_t | y_{1:t}) = \mathbb{E}((x_t - \hat{x}_{t|t})(x_t - \hat{x}_{t|t})^T | y_{1:t}). \quad \text{(II.15)}
\]

These expectations are calculated via a set of recursive equations [24, 26], for \(t > 2\),

\[
\hat{V}_{t|t-1} = A\hat{V}_{t-1|t-1}A^T + Q, \quad \text{(II.16)}
\]
\[
K_t = \hat{V}_{t|t-1}C^T (C\hat{V}_{t|t-1}C^T + R)^{-1}, \quad \text{(II.17)}
\]
\[
\hat{V}_{t|t} = \hat{V}_{t|t-1} - K_tC\hat{V}_{t|t-1}, \quad \text{(II.18)}
\]
\[
\hat{x}_{t|t-1} = A\hat{x}_{t-1|t-1}, \quad \text{(II.19)}
\]
\[
\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(y_t - C\hat{x}_{t|t-1} - \bar{y}), \quad \text{(II.20)}
\]

with initial conditions \(\hat{x}_{1|0} = \mu\) and \(\hat{V}_{1|0} = S\). The mean and covariance of the observed variable \(y_t\), conditioned on the previous observations \(y_{1:t-1}\), are computed from the conditional state expectations

\[
\hat{y}_{t|t-1} = \mathbb{E}(y_t | y_{1:t-1}) = C\hat{x}_{t|t-1} + \bar{y}, \quad \text{(II.21)}
\]
\[
\hat{\Sigma}_{t|t-1} = \text{cov}(y_t | y_{1:t-1}) = C\hat{V}_{t|t-1}C^T + R. \quad \text{(II.22)}
\]
Hence, the data log-likelihood can be computed efficiently using the “innovations” form [27]

\[
\log p(y_{1:\tau}) = \sum_{t=1}^{\tau} \log p(y_t|y_{1:t-1})
\]

(II.23)

\[
= \sum_{t=1}^{\tau} \log G(y_t, \hat{y}_{t|t-1}, \hat{\Sigma}_{t|t-1})
\]

(II.24)

\[
= \sum_{t=1}^{\tau} \log G(y_t, C\hat{x}_{t|t-1} + \bar{y}, C\hat{V}_{t|t-1}C^T + R).
\]

(II.25)

If \( R \) is an i.i.d. or diagonal covariance matrix (e.g. \( R = rI_m \)), then the steps of the Kalman filter can be computed efficiently using the matrix inversion lemma.

**II.C.2 Kalman smoothing filter**

The Kalman smoothing filter [27, 25] estimates the mean and covariance of the state \( x_t \) of an LDS, conditioned on the *entire* observed sequence \( y_{1:\tau} = (y_1, \ldots, y_\tau) \). Define the conditional expectations

\[
\hat{x}_{t|\tau} = \mathbb{E}(x_t|y_{1:\tau}),
\]

(II.26)

\[
\hat{V}_{t|\tau} = \text{cov}(x_t|y_{1:\tau}) = \mathbb{E}((x_t - \hat{x}_{t|\tau})(x_t - \hat{x}_{t|\tau})^T|y_{1:\tau}),
\]

(II.27)

\[
\hat{V}_{t,t-1|\tau} = \text{cov}(x_t, x_{t-1}|y_{1:\tau}) = \mathbb{E}((x_t - \hat{x}_{t|\tau})(x_{t-1} - \hat{x}_{t-1|\tau})^T|y_{1:\tau}).
\]

(II.28)

The expectations \( \hat{x}_{t|\tau} \) and \( \hat{V}_{t|\tau} \) are obtained with the backward recursions, for \( t = \tau, \ldots, 2 \)

\[
J_{t-1} = \hat{V}_{t-1|t-1}A^T(\hat{V}_{t|t-1})^{-1},
\]

(II.29)

\[
\hat{x}_{t-1|\tau} = \hat{x}_{t-1|t-1} + J_{t-1}(\hat{x}_{t|\tau} - A\hat{x}_{t-1|t-1}),
\]

(II.30)

\[
\hat{V}_{t-1|\tau} = \hat{V}_{t-1|t-1} + J_{t-1}(\hat{V}_{t|\tau} - \hat{V}_{t|t-1})J_{t-1}^T.
\]

(II.31)

The conditional covariance \( \hat{V}_{t,t-1|\tau} \) is computed recursively, for \( t = \tau, \ldots, 2 \)

\[
\hat{V}_{t-1,t-2|\tau} = \hat{V}_{t-1|t-1}J_{t-2}^T + J_{t-1}(\hat{V}_{t,t-1|\tau} - A\hat{V}_{t-1|t-1})J_{t-2}^T
\]

(II.32)

with initial condition \( \hat{V}_{\tau,\tau-1|\tau} = (I - K_\tau C)A\hat{V}_{\tau-1|\tau-1}. \)
II.D Parameter estimation

A number of methods are available to learn the parameters of the dynamic texture from a training video sequence $y_{1:\tau} = (y_1, \ldots, y_\tau)$, including maximum-likelihood methods (e.g. expectation-maximization [27]), non-iterative subspace methods (e.g. N4SID [28], CCA [29, 30]) or a suboptimal, but computationally efficient, least-squares procedure [13].

In this section, we review the expectation-maximization (EM) algorithm for learning the parameters of a dynamic texture, which is related to the EM algorithms derived for the mixture of dynamic textures and layered dynamic textures presented in Chapter IV and Chapter V. We also present the suboptimal least-squares procedure proposed in [13]. We then derive a set of sufficient statistics for the efficient on-line implementation of the suboptimal least-squares algorithm. The resulting on-line algorithm provides insight into the relationship between EM and the least-squares procedure of [13].

Note that, under some conditions\footnote{One of these conditions is that the parameter $n$ is selected to be the true state-space dimension!}, the subspace method CCA is also a maximum-likelihood estimator [30]. However, the large dimensionality of the image frames makes this method, and other non-iterative methods, infeasible for learning dynamic texture models. These methods require regressing between windows of the past and future observations, which is computed via a singular value decomposition (SVD) of a matrix with dimensions $(dm) \times \tau$, where $d$ is the length of the window. For CCA, typically $8 \leq d \leq 15$, and hence the SVD becomes infeasible when $m$ is large.

II.D.1 Expectation maximization

Given a $\tau$-length video sequence $y_{1:\tau} = (y_1, \ldots, y_\tau)$, we would like to learn the parameters $\Theta$ of a dynamic texture that best fits the data in the maximum-
likelihood sense [24], i.e.,

\[
\Theta^* = \arg\max_{\Theta} \log p(y_{1:\tau}; \Theta).
\] (II.33)

When the probability distribution depends on hidden variables (i.e. the output of the system is observed, but its state is unknown), the maximum-likelihood solution can be found with the EM algorithm [31]. For the dynamic texture, the observed information is the video sequence \(y_{1:\tau}\), and the missing data consists of the hidden state sequence \(x_{1:\tau}\) that produces \(y_{1:\tau}\). The EM algorithm is an iterative procedure that alternates between estimating the missing information with the current parameters, and computing new parameters given the estimate of the missing information. In particular, each iteration consists of

\[
E - \text{Step : } Q(\Theta; \hat{\Theta}) = \mathbb{E}_{X|Y; \hat{\Theta}}(\log p(X, Y; \Theta)),
\] (II.34)

\[
M - \text{Step : } \hat{\Theta}^* = \arg\max_{\Theta} Q(\Theta; \hat{\Theta}),
\] (II.35)

where \(p(X, Y; \Theta)\) is the complete-data likelihood of the observed and hidden state sequences, parameterized by \(\Theta\).

The EM algorithm for a dynamic texture (or LDS) was derived in [27, 32, 33], and is presented in Algorithm 1. We assume that the observed sequence \(y_{1:\tau}\) has zero mean, although the algorithm could be trivially extended to the case of non-zero mean. In the E-step, the Kalman smoothing filter is used to compute the second moments of the state-space, conditioned on the observed sequence \(y_{1:\tau}\), as in (II.36). Next, the state-space statistics are aggregated over time in (II.37). In the M-step, the parameters of the DT are recomputed according to (II.38).

**II.D.2 Least-squares**

The dynamic texture parameters are frequently learned with a least-squares algorithm proposed in [13], which learns the spatial and temporal parameters of the model separately. Given an observed video sequence (in matrix
Algorithm 1 EM for dynamic textures

Input: observed sequence $y_{1:\tau}$.

Initialize $\Theta = \{A, Q, C, R, \mu, S\}$.

repeat

{Expectation Step}

Use the Kalman smoothing filter (Section II.C) to compute conditional expectations:

\begin{align*}
\hat{x}_t &= \mathbb{E}[x_t|y_{1:\tau}], \\
\hat{P}_{t,t} &= \mathbb{E}[x_t x_T^t|y_{1:\tau}] = \text{cov}(x_t|y_{1:\tau}) + \hat{x}_t \hat{x}_t^T, \\
\hat{P}_{t,t-1} &= \mathbb{E}[x_t x_{t-1}^T|y_{1:\tau}] = \text{cov}(x_t, x_{t-1}|y_{1:\tau}) + \hat{x}_t \hat{x}_{t-1}^T. 
\end{align*}

Compute aggregated expectations:

\begin{align*}
\Phi &= \sum_{t=1}^{\tau} \hat{P}_{t,t}, \quad \varphi = \sum_{t=2}^{\tau} \hat{P}_{t,t}, \quad \phi = \sum_{t=2}^{\tau} \hat{P}_{t-1,t-1}, \\
\Psi &= \sum_{t=2}^{\tau} \hat{P}_{t,t-1}, \quad \Lambda = \sum_{t=1}^{\tau} y_t y_t^T, \quad \Gamma = \sum_{t=1}^{\tau} y_t (\hat{x}_t)^T. 
\end{align*}

{Maximization Step}

Compute new parameters:

\begin{align*}
C^* &= \Gamma \Phi^{-1}, \quad R^* = \Lambda - C^* \Gamma^T, \quad \mu_j^* = \hat{x}_1, \\
A^* &= \Psi \phi^{-1}, \quad Q^* = \frac{1}{\tau-1} \left( \varphi - A^* \Psi^T \right), \quad S^* = \text{cov}(x_1|y_{1:\tau}). 
\end{align*}

until convergence

Output: $\Theta = \{A, Q, C, R, \mu, S\}$. 
form), $Y_{1:τ} = [y_1 \cdots y_τ] \in \mathbb{R}^{m \times τ}$, the mean is first estimated by the sample mean

$$\bar{y} = \frac{1}{τ} \sum_{t=1}^{τ} y_t,$$  \hspace{1cm} (II.39)

and the mean-subtracted video sequence

$$\tilde{Y}_{1:τ} = [\tilde{y}_1 \cdots \tilde{y}_τ],$$  \hspace{1cm} (II.40)

where $\tilde{y}_t = y_t - \bar{y}$, $\forall t$, is used in all subsequent computations.

To estimate the model parameters, the video sequence is subject to a principal component analysis (PCA), performed with recourse to the singular value decomposition (SVD)

$$\tilde{Y}_{1:τ} = USV^T.$$  \hspace{1cm} (II.41)

The observation matrix is estimated from the $n$ principal components of largest eigenvalue

$$C = [u_1 \cdots u_n],$$  \hspace{1cm} (II.42)

where $u_i$ is the $i$-th column of $U$, and it was assumed that the diagonal entries of $S$ are ordered by decreasing value. The state-space variables are then estimated with

$$\hat{X}_{1:τ} = [\hat{x}_1 \cdots \hat{x}_τ] = CT\tilde{Y}_{1:τ},$$  \hspace{1cm} (II.43)

leading to the least square estimate of the state-transition matrix

$$A = \hat{X}_{2:τ}(\hat{X}_{1:τ-1})^\dagger,$$  \hspace{1cm} (II.44)

where $X^\dagger = X^T(XX^T)^{-1}$ is the Moore-Penrose pseudo-inverse of $X$. The state noise is estimated from the state-prediction residual error

$$\hat{V}_{1:τ-1} = \hat{X}_{2:τ} - A\hat{X}_{1:τ-1},$$  \hspace{1cm} (II.45)

$$Q = \frac{1}{τ-1} \hat{V}_{1:τ-1}(\hat{V}_{1:τ-1})^T,$$  \hspace{1cm} (II.46)
and the initial state is assumed as
\[ \mu = \hat{x}_1, \quad S = Q. \]  
(II.47)

Finally, the observation noise is estimated from the reconstruction error
\[ \hat{W}_{1:T} = \hat{Y}_{1:T} - C\hat{X}_{1:T}, \]  
(II.48)
\[ R = \frac{1}{\tau} \hat{W}_{1:T} \hat{W}_{1:T}^T. \]  
(II.49)

If the observation noise covariance is assumed i.i.d., i.e. \( R' = rI_m \), then the variance estimate \( r \) is the mean of the diagonal elements of the full covariance estimate \( R \), i.e. \( r = \frac{1}{m} \sum_{i=1}^{m} [R]_{i,i} \).

The least-squares method is summarized in Algorithm 2. Although sub-optimal in the maximum-likelihood sense, this procedure has been shown to lead to good estimates of dynamic texture parameters in various applications, including video synthesis [13] and recognition [16]. In the next section, we re-derive the least-squares method using sufficient statistics. This will lead to an efficient on-line algorithm, and show an interesting connection between the least-squares algorithm and EM.

**II.D.3 On-line estimation with least-squares**

In this section, we modify the least-squares algorithm, presented in the previous section, for on-line learning. In particular, we desire to update a dynamic texture \( \Theta \) learned from a video sequence \( y_{1:T-1} \) with a new observed video frame \( y_T \). The estimates of the state-space variables \( \hat{X}_{1:T} \) play a central role in the least-squares algorithm. However, because 1) they depend on the principal component basis (through the matrix \( C \)), and 2) this basis changes with each new observation, they are not suitable sufficient statistics for on-line parameter estimation. An on-line version of the least-squares procedure requires an alternative set of sufficient statistics which do not depend on \( C \). We next define these statistics,

\[ \hat{\Phi} = \hat{Y}_{1:T} \hat{Y}_{1:T}^T, \quad \hat{\varphi} = \hat{Y}_{2:T} \hat{Y}_{2:T}^T, \quad \hat{\phi} = \hat{Y}_{1:T-1} \hat{Y}_{1:T-1}^T, \]
\[ \hat{\psi} = \hat{Y}_{2:T} \hat{Y}_{1:T-1}^T, \quad \hat{\eta} = \hat{y}_1 \hat{y}_1^T, \quad \hat{\xi} = \hat{y}_1, \]  
(II.51)
Algorithm 2 Least-squares for dynamic textures

**Input:** observed sequence $y_{1:T}$.

Compute sample mean: $\bar{y} = \frac{1}{T} \sum_{t=1}^{T} y_{t}$.

Subtract mean: $\tilde{y}_{t} = y_{t} - \bar{y}, \forall t$, $\tilde{Y}_{1:T} = [\tilde{y}_{1} \cdots \tilde{y}_{T}]$.

Compute SVD: $\tilde{Y}_{1:T} = USV^T$.

Estimate observation matrix: $C = [u_1 \cdots u_n]$.

Estimate state-space variables: $\hat{X}_{1:T} = [\hat{x}_1 \cdots \hat{x}_T] = C^T \tilde{Y}_{1:T}$.

Estimate remaining parameters

$A = \hat{X}_{2:T}(\hat{X}_{1:T-1})^\dagger$, $\mu = \hat{x}_1$, $S = Q$,

$\hat{V}_{1:T-1} = \hat{X}_{2:T} - A\hat{X}_{1:T-1}$, $Q = \frac{1}{T-1} \hat{V}_{1:T-1}(\hat{V}_{1:T-1})^T$, (II.50)

$\hat{W}_{1:T} = \hat{Y}_{1:T} - C\hat{X}_{1:T}$, $R = \frac{1}{T-1} \hat{W}_{1:T} \hat{W}_{1:T}^T$. (II.51)

**Output:** $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$.

and will show how they are derived in the remainder of this section. We start by substituting (II.43) into (II.44), to rewrite the estimate of the state-transition matrix as

$$A = (C^T \hat{Y}_{2:T})(C^T \hat{Y}_{1:T-1})^\dagger \quad \text{(II.52)}$$

$$= (C^T \hat{Y}_{2:T} \hat{Y}_{1:T-1} C)(C^T \hat{Y}_{1:T-1} \hat{Y}_{1:T-1} C)^{-1} \quad \text{(II.53)}$$

$$= (C^T \hat{\psi} C)(C^T \hat{\phi} C)^{-1}. \quad \text{(II.54)}$$

Note that this is an estimate of $A$ at time $\tau$, which only depends on the estimate $C$ of the PCA basis at this time step, and the sufficient statistics $\hat{\psi}$ and $\hat{\phi}$. The latter can be updated recursively, e.g.

$$\hat{\psi}^{(\tau)} = \hat{Y}_{2:T} \hat{Y}_{1:T-1}^T \quad \text{(II.55)}$$

$$= \sum_{t=2}^{\tau} \tilde{y}_{t} \tilde{y}_{t-1}^T = \sum_{t=2}^{\tau-1} \tilde{y}_{t} \tilde{y}_{t-1}^T + \tilde{y}_{\tau} \tilde{y}_{\tau-1}^T \quad \text{(II.56)}$$

$$= \hat{\psi}^{(\tau-1)} + \bar{y}_{\tau} \bar{y}_{\tau-1}^T. \quad \text{(II.57)}$$

where the superscript indicates the time-step to which the statistic refers. Similar recursive formulas can be derived for all statistics of (II.51), and are summarized
in (II.70). The PCA basis can be estimated in a similar form. It suffices to note
that the SVD computation of (II.42) is equivalent to computing the \( n \) principal
components of \( \hat{\Phi} \), as defined in (II.51). Hence \( \hat{C} \) can be estimated as
\[
\hat{C} = \text{pca}(\hat{\Phi}, n) \tag{II.58}
\]
where pca(\( \Sigma, n \)) returns the top \( n \) principal components of \( \Sigma \), and where \( \hat{\Phi} \) is
updated recursively, using (II.70).

To derive an on-line estimate of the covariance \( Q \), we note that
\[
Q = \frac{1}{\tau - 1} (\hat{X}_{2: \tau} - A\hat{X}_{1: \tau - 1})(\hat{X}_{2: \tau} - A\hat{X}_{1: \tau - 1})^T \tag{II.59}
\]
\[
= \frac{1}{\tau - 1} (\hat{X}_{2: \tau} - A\hat{X}_{1: \tau - 1})\hat{X}_{2: \tau}^T
\]
\[
\quad - \hat{X}_{2: \tau}\hat{X}_{1: \tau - 1}A^T + A\hat{X}_{1: \tau - 1}\hat{X}_{1: \tau - 1}A^T) \tag{II.60}
\]
\[
= \frac{1}{\tau - 1} (\hat{X}_{2: \tau}^T\hat{X}_{2: \tau} - A\hat{X}_{1: \tau - 1}\hat{X}_{2: \tau}) \tag{II.61}
\]
\[
= \frac{1}{\tau - 1} (C^T\hat{Y}_{2: \tau}\hat{Y}_{2: \tau} - AC^T\hat{Y}_{1: \tau - 1}\hat{Y}_{2: \tau}) \tag{II.62}
\]
\[
= \frac{1}{\tau - 1} (C^T\hat{\varphi}C - AC^T\hat{\psi}C) \tag{II.63}
\]
where we have used, in (II.60),
\[
A\hat{X}_{1: \tau - 1}\hat{X}_{1: \tau - 1}A^T = (\hat{X}_{2: \tau}\hat{X}_{1: \tau - 1}^*)\hat{X}_{1: \tau - 1}\hat{X}_{1: \tau - 1}A^T = \hat{X}_{2: \tau}\hat{X}_{1: \tau - 1}A^T.
\]
The mean and variance of the initial state are computed via
\[
\hat{\mu} = \hat{x}_1 = \hat{C}^T\tilde{y}_1 = \hat{C}^T\hat{\xi}, \tag{II.64}
\]
\[
\hat{S} = (\hat{x}_1 - \hat{\mu})(\hat{x}_1 - \hat{\mu})^T = \hat{C}^T\tilde{y}_1\tilde{y}_1^T\hat{C} - \hat{\mu}\hat{\mu}^T = (\hat{C}^T\hat{\eta}\hat{C}) - \hat{\mu}\hat{\mu}^T. \tag{II.65}
\]
Finally, the observation noise is estimated from the reconstruction error \( \hat{Y}_{1: \tau} - C\hat{X}_{1: \tau} \),
\[
R = \frac{1}{\tau} (\hat{Y}_{1: \tau} - C\hat{X}_{1: \tau})(\hat{Y}_{1: \tau} - C\hat{X}_{1: \tau})^T \tag{II.66}
\]
\[
= \frac{1}{\tau} (\hat{Y}_{1: \tau} - C\hat{C}^T\hat{Y}_{1: \tau})(\hat{Y}_{1: \tau} - C\hat{C}^T\hat{Y}_{1: \tau})^T \tag{II.67}
\]
\[
= \frac{1}{\tau} (I - CC^T)\hat{Y}_{1: \tau}\hat{Y}_{1: \tau}^T(I - CC^T) \tag{II.68}
\]
\[
= (I - CC^T)\hat{\Phi}(I - CC^T). \tag{II.69}
\]
The on-line least-squares algorithm is summarized in Algorithm 3. The initial conditions are \( \Phi^{(0)} = \Psi^{(0)} = 0 \), with \( \hat{\eta} \) and \( \hat{\xi} \) is defined in (II.51). Note that the on-line estimates obtained with these recursive sufficient statistics are identical to the solution produced by the least-squares algorithm of Section II.D.2, in the zero-mean case. The on-line algorithm will be used in Chapter VII to perform adaptive background modeling using dynamic textures, although the algorithm is suitable for any application of dynamic textures that requires on-line video processing.

**Algorithm 3 On-line least-squares for dynamic textures**

**Input:** new observation \( y_{\tau} \), DT parameters \( \Theta \), sufficient statistics \( \{ \hat{\Phi}^{(\tau-1)}, \hat{\phi}^{(\tau-1)}, \hat{\psi}^{(\tau-1)}, \hat{\eta}, \hat{\xi} \} \).

Update mean: \( \bar{y} \leftarrow \frac{\tau-1}{\tau} \bar{y} + \frac{1}{\tau} y_{\tau} \).

Subtract mean: \( \check{y}_{\tau} = y_{\tau} - \bar{y} \).

Update statistics:

\[
\hat{\Phi}^{(\tau)} = \hat{\Phi}^{(\tau-1)} + \check{y}_{\tau} \check{y}_{\tau}^T, \quad \hat{\phi}^{(\tau)} = \hat{\Phi}^{(\tau)} - \hat{\eta},
\]

\( \hat{\Phi}^{(\tau)} = \hat{\Phi}^{(\tau-1)} \), \( \hat{\psi}^{(\tau)} = \hat{\Psi}^{(\tau-1)} + \check{y}_{\tau} \check{y}_{\tau-1}^T \).

Estimate observation matrix: \( C = \text{pca}(\hat{\Phi}, n) \).

Recompute parameters:

\[
A = (C^T \hat{\psi}C)(C^T \hat{\phi}C)^{-1}, \quad \mu = C^T \hat{\xi}, \quad S = C^T \hat{\eta}C - \mu \mu^T,
\]

\[
Q = \frac{1}{\tau-1}(C^T \hat{\psi}C - AC^T \hat{\psi}TC), \quad R = (I - CC^T)\hat{\Phi}(I - CC^T).
\]

**Output:** \( \Theta = \{ A, Q, C, R, \mu, S, \bar{y} \} \), new statistics \( \{ \hat{\Phi}^{(\tau)}, \hat{\phi}^{(\tau)}, \hat{\phi}^{(\tau)}, \hat{\psi}^{(\tau)}, \hat{\eta}, \hat{\xi} \} \).

Rewriting the least-squares solution in terms of sufficient statistics also exposes an interesting connection between the least-squares algorithm and the EM algorithm for dynamic textures. In particular, the parameter estimates of (II.71) utilize matrices of the form \( C^T \phi C \), which is equivalent to the projection of the image statistic \( \phi \) into the state-space of the LDS. These projections can be viewed as approximations to the aggregated statistics computed in the E-step of the EM algorithm.
algorithm in (II.36, II.37):

\[
\begin{align*}
CT\hat{\phi}C & \approx \sum_{t=2}^{T} \mathbb{E}(x_{t-1}x_{t-1}^T|y_{1:t}), \\
CT\hat{\psi}C & \approx \sum_{t=2}^{T} \mathbb{E}(x_tx_t^T|y_{1:t}), \\
CT\hat{\psi}C & \approx \sum_{t=2}^{T} \mathbb{E}(x_t|x_{t-1}^T|y_{1:t}).
\end{align*}
\tag{II.72}
\]

Under this interpretation, the estimates for \(A, Q, S,\) and \(\mu\) in (II.71) are equivalent to those in the M-step of the EM algorithm in (II.38). Hence, the least-squares solution can be viewed as a single iteration of the EM algorithm, where \(C\) is approximated as the PCA basis of the observations, and the E-step is approximated by projecting the image statistics into the state-space. In practice, the least-squares estimate serves as a good initialization for the EM algorithm, which typically converges after a few iterations.

Finally, the proposed on-line procedure can also be used to estimate the parameters of a dynamic texture from \textit{multiple} video samples. In particular, given a set of \(N\) mean-subtracted videos \(\{\tilde{Y}_{1:t}^{(1)}, \ldots, \tilde{Y}_{1:t}^{(N)}\}\), the sufficient statistics are computed by averaging over all samples, e.g.

\[
\hat{\Phi} = \frac{1}{N} \sum_{i=1}^{N} \tilde{Y}_{1:t}^{(i)} (\tilde{Y}_{1:t}^{(i)})^T, \tag{II.73}
\]

and similarly for the other statistics. The parameters are then estimated with (II.71).

\section*{II.E Acknowledgements}

The text of Section II.D.3 is based on the material as it appears in: A. B. Chan, V. Mahadevan, and N. Vasconcelos, “Generalized Stauffer-Grimson background subtraction for dynamic scenes”, submitted for publication to Machine Vision and Applications. The dissertation author was a primary researcher and an author of the cited material.
Chapter III

Probabilistic kernels for the classification of dynamic textures
III.A Introduction

In the context of classification, detection, and recognition problems, the probabilistic representation has various properties that are known to be assets for perception [34], e.g. existence of principled inference formalisms that allow the fusion of diverse sources of information, the ability to incorporate domain-knowledge in the form of prior beliefs, etc. There are, nevertheless, core aspects in which it also has strong shortcomings. In particular, while it can lead to optimal classifiers by simple application of Bayesian decision theory, these classifiers have weak generalization guarantees, and can be quite sensitive to the dimensionality of the underlying feature space, or prone to over-fitting when the models have large numbers of parameters. This is a source of particular concern for the problems involving dynamic textures, since spatio-temporal autoregressive modeling tends to require high dimensional feature and state spaces.

An alternative classification framework [35], which delivers large-margin classifiers of much better generalization ability (e.g. the now popular support vector machine), does exist but has strong limitations of its own. For the classification of spatio-temporal data-streams, the most restrictive among these is a requirement for the representation of those data-streams as points in Euclidean space. These points are then mapped into a high-dimensional feature space by a kernel function that transforms Euclidean distances in domain space into distances defined over a manifold embedded in range space. The Euclidean representation is particularly troublesome for spatio-temporal processes, where different instances of a process may have different temporal extents (e.g. two similar video streams with different numbers of frames), or be subject to simple transformations that are clearly irrelevant for perception and classification (e.g. a change of sampling rate), but can map the same data-stream into very different points of Euclidean space.

Recent developments in the area of probabilistic kernels have shown significant promise to overcome these limitations. Probabilistic kernels are kernels
that act on pairs of generative probabilistic models, enabling simultaneous support for complex statistical inference, which is characteristic of probabilistic representations, and good generalization guarantees, which are characteristic of large-margin learning. Although the feasibility of applying these kernels to vision problems has been demonstrated on relatively simple recognition tasks, e.g. the recognition of objects presented against a neutral background [36], we believe that this is unsatisfactory in two ways. First, the greatest potential for impact of probabilistic kernels is in the solution of classification problems where 1) simple application of Bayesian decision theory is likely to fail, e.g. problems involving large state-space models and high-dimensional features, and 2) the inappropriateness of the Euclidean representation makes the traditional large-margin solutions infeasible. Second, many of the recognition problems for which there are currently no good solutions in the vision literature, e.g. those involving visual processes modeled as dynamic textures, are exactly of this type.

Both of these points are addressed in this chapter, which makes contributions at two levels. On one hand, we introduce a procedure for the design of large-margin classifiers for dynamic textures. This includes the derivation of a discriminant distance function (the Kullback-Leibler divergence) for dynamic textures and its application to the design of probabilistic kernels. On the other, we demonstrate the practical feasibility of large margin classification for vision problems involving complex spatio-temporal visual stimuli, such as the classification of video textures or the classification of patterns of highway traffic flow under variable environmental conditions. The new large-margin solution is shown to perform well above the state of the art and to produce quite promising results for difficult problems, such as monitoring highway congestion.

The remainder of this chapter is organized as follows. In Section III.B we describe the framework combining probabilistic kernels with support vector machines, and in Section III.C we derive the Kullback-Leibler divergence between dynamic textures, enabling the extension of probabilistic kernels to dynamic tex-
In Section III.D, we review the Martin distance between dynamic textures, and finally, in Section III.E we present results on video retrieval and classification using the probabilistic kernel.

III.B Support vector machines and probabilistic kernels

A support vector machine (SVM) [35] is a discriminative classifier that constructs a maximum-margin hyperplane between two classes using a set of training examples \( \{x_1, \ldots, x_N\} \in \mathcal{X} \). The SVM provides strong generalization guarantees for learning and usually leads to improved performance, outside the training set, when compared to classical methods based on Bayesian decision theory [37]. The training examples that are most difficult to classify are referred to as support vectors, and determine the separating hyperplane.

The SVM can be augmented by using the “kernel” trick, which maps the training examples into a high-dimensional non-linear feature space. This feature space transformation is defined by the kernel function \( k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \). One interpretation of the kernel function is that \( k(x_i, x_j) \) measures the similarity between the two points \( x_i \) and \( x_j \) in the space \( \mathcal{X} \). A popular example is the Gaussian kernel, defined as \( k_g(x_i, x_j) = \exp(-\gamma \| x_i - x_j \|^2) \).

If the training examples are represented as probabilistic models (e.g. dynamic textures), the kernel becomes a measure of similarity between probability distributions. A probabilistic kernel is thus defined as a mapping \( k : \mathcal{P} \times \mathcal{P} \rightarrow \mathbb{R} \), where \( \mathcal{P} \) is the space of probability distributions. One such kernel is the Kullback-Leibler kernel [38], defined as

\[
k_{KL}(p, q) = e^{-\gamma (D(p\|q) + D(q\|p))} \tag{III.1}
\]

where \( D(p\|q) \) is the Kullback-Leibler (KL) divergence between the probability distributions \( p(x) \) and \( q(x) \) [39]

\[
D(p\|q) = \int_{\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \, dx. \tag{III.2}
\]
The KL divergence is a natural distance measure between two probability distributions, and the KL kernel in probability space is analogous to the Gaussian kernel in Euclidean space. The KL kernel has been shown to achieve very good results in the domains of object [36] and speech [38] recognition.

III.C Kullback-Leibler divergence

In this section we introduce a probabilistic kernel for visual processes that can be modeled as dynamic textures. The dynamic texture model provides a probability distribution of the texture in both image and state spaces. This allows the derivation of two kernels that can ground the classification in either the appearance or the flow of the dynamic texture. Grounding the classification on image space tends to favor iconic pixel matches and leads to best performance when the goal is to differentiate between dynamic textures of different visual appearance (e.g. a flock of birds from a school of fish in Figure I.1). Under this approach, two sequences of distinct textures subject to similar motion can be correctly identified. It is, however, not clear that the dynamic texture model is of great significance in this context: a simple appearance classifier based, for example, on the principal component decomposition of the sequences may achieve good results. Ideally, the kernel based on the dynamic texture model should achieve performance at least as good as that of a static kernel, when this is the case.

An alternative classification scenario is that where the different classes have similar appearance and all the discriminant information is contained in the motion flow. For example, problems such as determining the level of traffic on a highway, or detecting outliers and unusual events (e.g. cars speeding or committing other traffic violations). Since for these cases the iconic pixel matching inherent to existing static kernels is clearly inappropriate, these are the cases where dynamic texture kernels have the greatest potential for improvement over the state of the art. In summary, depending on the specific classification problem, it may be advisable
to ground the classification on either the state space or the image space components of the dynamic texture model.

The asymptotic KL divergence rate between two random processes with distributions \( p(X) \) and \( q(X) \), over random variables \( X = (x_1, x_2, x_3, \ldots) \), is given by

\[
D(p(X) \| q(X)) = \lim_{\tau \to \infty} D_{\tau}(p(x_{1:\tau}) \| q(x_{1:\tau})), \tag{III.3}
\]

where \( D_{\tau}(p(x_{1:\tau}) \| q(x_{1:\tau})) \) is the KL divergence rate between \( p \) and \( q \) over the sub-sequence \( x_{1:\tau} = (x_1, \ldots, x_\tau) \), i.e.

\[
D_{\tau}(p(x_{1:\tau}) \| q(x_{1:\tau})) = \frac{1}{\tau} \int p(x_{1:\tau}) \log \frac{p(x_{1:\tau})}{q(x_{1:\tau})} dx_{1:\tau}. \tag{III.4}
\]

In the remainder of this section we derive the KL divergence rate between two dynamic textures, parameterized by \( \Theta_p = \{A_p, Q_p, C_p, R_p, \mu_p, S_p, \bar{y}_p\} \) and \( \Theta_q = \{A_q, Q_q, C_q, R_q, \mu_q, S_q, \bar{y}_q\} \), in both the state space and the image space.

### III.C.1 Projection between state spaces

The KL divergence between state spaces cannot be computed directly because each dynamic texture uses a different hidden-state space (defined by \( C_p \) and \( C_q \)). Instead, one state space must be projected into the other by applying a sequence of two transformations: 1) from the original state space into image space, and 2) from image space into the target state space. If the original state space is that of \( x_t^{(p)} \) and the target that of \( x_t^{(q)} \), this is the transformation \( \hat{x}_t^{(p)} = Fx_t^{(p)} \) with \( F = C_q^T C_p \) (or \( F = (C_q^T C_q)^{-1}C_q^T C_p \) if \( C_q \) is not orthogonal). From (II.1),

\[
x_t^{(p)} = \quad A_p x_t^{(p)} + v_t, \tag{III.5}
\]

\[
F x_t^{(p)} = \quad F A_p F^{-1} x_t^{(p)} + F v_t, \tag{III.6}
\]

\[
\hat{x}_t^{(p)} = \quad \hat{A}_p \hat{x}_t^{(p)} + \hat{v}_t, \tag{III.7}
\]

where the transformed nose \( \hat{v}_t \) has covariance \( \hat{Q} \). Hence, the transformation \( F \) of a state-process with parameters \( \{A_p, Q_p, \mu_p, S_p\} \) is a state-process with parameters
\{\hat{A}_p, \hat{Q}_p, \hat{\mu}_p, \hat{S}_p\}$, computed according to
\[
\hat{A}_p = FA_pF^{-1}, \quad \hat{Q}_p = FQ_pF^T, \quad \hat{\mu}_p = F\mu_p, \quad \hat{S}_p = FS_pF^T. \tag{III.8}
\]
The KL divergence between state spaces can now be computed with this transformed state model.

### III.C.2 KL divergence between state spaces

We first derive the KL divergence rate between the probability distributions of the hidden states of two dynamic textures, and then take the limit as $\tau \to \infty$ to obtain the asymptotic KL divergence rate.

**Theorem 1** (KL divergence rate between state spaces). The KL divergence rate between the state-space distributions $p(x_{1:\tau})$ and $q(x_{1:\tau})$ of two dynamic textures parameterized by $\Theta_p$ and $\Theta_q$, respectively, is given by
\[
D_\tau(p(x_{1:\tau}) \parallel q(x_{1:\tau})) = \frac{1}{2\tau} \left[||\mu_p - \mu_q||^2_{S_2} + \log \frac{|S_q|}{|S_p|} + \text{tr}(S_q^{-1}S_p)\right] - \frac{n}{2} \tag{III.9}
\]
\[
+ \frac{1}{2} \text{tr} \left(\hat{A}^TQ_q^{-1}\hat{A} - \frac{1}{\tau} \sum_{t=1}^{\tau-1} (S_t + \mu_t\mu_t^T)\right)
+ \frac{\tau - 1}{2\tau} \left[\log \frac{|Q_q|}{|Q_p|} + \text{tr}(Q_q^{-1}Q_p)\right],
\]
where $\hat{A} = A_p - A_q$, and $S_t$ and $\mu_t$ are the covariance and mean of the state $x_t$ of the first dynamic texture $\Theta_p$, as calculated in (II.8).

**Proof.** Using the chain rule of divergence [40] and the fact that the state sequence is a first-order Markov process, the KL divergence between $p(x_{1:\tau})$ and $q(x_{1:\tau})$ becomes
\[
D(p(x_{1:\tau}) \parallel q(x_{1:\tau})) = D(p(x_1) \parallel q(x_1)) + \sum_{t=2}^{\tau} D(p(x_t|x_{t-1}) \parallel q(x_t|x_{t-1})), \tag{III.10}
\]
where the conditional KL divergence is defined as
\[
D(p(x|y) \parallel q(x|y)) = \int p(x, y) \log \frac{p(x|y)}{q(x|y)} dxdy. \tag{III.11}
\]
The initial state vectors are distributed as Gaussians, hence the KL divergence between the initial state distributions is
\[
D(p(x_1) \parallel q(x_1)) = \frac{1}{2} ||\mu_p - \mu_q||^2_{S_2} + \frac{1}{2} \log \frac{|S_q|}{|S_p|} + \frac{1}{2} \text{tr}(S_q^{-1}S_p) - \frac{n}{2} \tag{III.12}
\]
For the conditional KL terms, we have

\[
D(p(x_t|x_{t-1}) \| q(x_t|x_{t-1})) \\
= \int p(x_{t-1}) \int p(x_t|x_{t-1}) \frac{p(x_t|x_{t-1})}{q(x_t|x_{t-1})} dx_t dx_{t-1} \\
= \int p(x_{t-1}) \int G(x_t,A_px_{t-1},Q_p) \log \frac{G(x_t,A_px_{t-1},Q_p)}{G(x_t,A_qx_{t-1},Q_q)} dx_t dx_{t-1} \\
= \int p(x_{t-1}) \frac{1}{2} \left[ \| (A_p - A_q)x_{t-1} \|_Q^2 + \log \frac{|Q_q|}{|Q_p|} + \text{tr}(Q_q^{-1}Q_p) - n \right] dx_{t-1} \\
= \frac{1}{2} \left[ \text{tr}(\tilde{A}^T Q_q^{-1} \tilde{A} (S_{t-1} + \mu_{t-1} \mu_{t-1}^T)) + \log \frac{|Q_q|}{|Q_p|} + \text{tr}(Q_q^{-1}Q_p) - n \right], \tag{III.13}
\]

where \( \tilde{A} = A_p - A_q \), and in the last line we have used the following property:

\[
\mathbb{E}(\| Ax \|_B^2) = \mathbb{E}(x^T A^T B^{-1} A x) = \mathbb{E}(\text{tr}(A^T B^{-1} A x x^T)) = \text{tr}(A^T B^{-1} A (\Sigma + \mu \mu^T))
\]

where \( \mu \) and \( \Sigma \) are the mean and covariance of the distribution of \( x \). Finally, (III.9) is obtained by substituting (III.12) and (III.13) into (III.10).

**Theorem 2** (Asymptotic KL divergence rate between state spaces). **Consider two dynamic textures parameterized by \( \Theta_p \) and \( \Theta_q \). If \( A_p \) is a stable transition matrix (i.e. all eigenvalues are within the unit circle), then the asymptotic KL divergence rate between the two state processes \( p(X) \) and \( q(X) \), where \( X = (x_1,x_2,x_3,...) \), is given by

\[
D(p(X) \| q(X)) = \frac{1}{2} \text{tr}(\tilde{A}^T Q_q^{-1} \tilde{A} S) + \frac{1}{2} \log \frac{|Q_q|}{|Q_p|} + \frac{1}{2} \text{tr}(Q_q^{-1} Q_p) - \frac{n}{2},
\]

where \( \tilde{A} = A_p - A_q \), and \( S \) is obtained by solving the Lyapunov equation \( S = A_p S A_p^T + Q_p \).

**Proof.** The asymptotic KL divergence rate is obtained by taking the limit of (III.9) as \( \tau \to \infty \). We first examine the steady-state values of \( \mu_t \) and \( S_t \) as \( t \to \infty \). Unwrapping the recursion of (II.8), we have

\[
\lim_{t \to \infty} \mu_t = \lim_{t \to \infty} A_p^{t-1} \mu_p = 0, \tag{III.14}
\]
where we have used \( \lim_{t \to \infty} A_t^p = 0 \), since \( A_p \) has eigenvalues within the unit circle. Similarly, unwrapping the recursion of (II.8), we have

\[
\lim_{t \to \infty} S_t = \lim_{t \to \infty} \left[ A_p^{-1} S_p (A_p^{-1})^T + \sum_{j=0}^{t-2} A_p^j Q_p (A_p^{-j})^T \right] = \sum_{j=0}^{\infty} A_p^j Q_p (A_p^{-j})^T = S. \tag{III.15}
\]

The sum over \( j \) converges and can be found by solving the Lyapunov equation \( S = A_p S A_p^T + Q_p \). Finally, by Cesàro’s mean (Theorem 4.2.3, [40]), we have

\[
\lim_{\tau \to \infty} \frac{1}{\tau} \sum_{t=1}^{\tau-1} (S_t + \mu_t \mu_t^T) = \lim_{t \to \infty} (S_t + \mu_t \mu_t^T) = S. \tag{III.16}
\]

Finally, substituting (III.17) into (III.9), and noting that the \( \lim_{\tau \to \infty} \frac{1}{\tau} = 0 \) and \( \lim_{\tau \to \infty} \frac{\tau-1}{\tau} = 1 \), yields (III.14).

**III.C.3 KL divergence in the image space**

In this section we derive an expression for the KL divergence rate and asymptotic KL divergence rate between two dynamic textures in the observation (image) space. The computation of the KL divergence rate is based on taking expectations of the conditional log-likelihood of the DT, which can be computed efficiently using the Kalman filter. The asymptotic KL divergence rate is computed by solving a set of Sylvester equations, after running the Kalman filters defined by the DT to steady-state. In previous work, the asymptotic KL divergence rate between stationary Gaussian processes (without memory) was derived using spectral density matrices [41, 42]. The asymptotic KL divergence rate between arbitrary time-invariant discrete Markov sources (without a hidden state) was also derived in [43]. In [44], an information measure (similar to the KL divergence) between LDS with zero mean was derived in the time-domain. In this work, we assume the general case, where the observation and state of the LDS both have non-zero mean.
Preliminaries

Before presenting the main results, we must first briefly touch on some properties of the Kalman filter. First, the recursive equations of the Kalman filter in (II.16) and (II.19) can be rewritten as:

\[ K_{t-1} = \hat{V}_{t-1|t-2}^{C}(C\hat{V}_{t-1|t-2}^{C} + R)^{-1}, \]  \hspace{1cm} (III.18)

\[ F_{t-1} = A - AK_{t-1}C, \]  \hspace{1cm} (III.19)

\[ \hat{V}_{t|t-1} = F_{t-1}\hat{V}_{t-1|t-2}A^T + Q, \]  \hspace{1cm} (III.20)

\[ \hat{x}_{t|t-1} = F_{t-1}\hat{x}_{t-1|t-2} + AK_{t-1}(y_{t-1} - \bar{y}). \]  \hspace{1cm} (III.21)

We also require some results on sensitivity analysis of the Kalman filter [26]. In particular, we are interested in expressions for the mean and variance of the conditional state estimator \( \hat{x}_{t|t-1} \) for two cases: 1) the observed sequence is generated by the same DT used for the Kalman filter (i.e. matched filter); and 2) the observed sequence is generated from a different DT than the Kalman filter (i.e. mismatched filter). Given two DT \( \Theta_p \) and \( \Theta_q \), with marginal statistics \( \{\mu_{t}^{(p)}, S_{t}^{(p)}, \gamma_{t}^{(p)}, \Sigma_{t}^{(p)}\} \) and \( \{\mu_{t}^{(q)}, S_{t}^{(q)}, \gamma_{t}^{(q)}, \Sigma_{t}^{(q)}\} \) computed by (II.8, II.9), and associated Kalman filters \( \{K_{t}^{(p)}, F_{t}^{(p)}, \hat{V}_{t|t-1}^{(p)}, \hat{x}_{t|t-1}^{(p)}\} \) and \( \{K_{t}^{(q)}, F_{t}^{(q)}, \hat{V}_{t|t-1}^{(q)}, \hat{x}_{t|t-1}^{(q)}\} \), we have the following lemmas (see Appendix III.H.1 for proofs):

**Lemma 1** (Matched filter). Suppose the observation \( y_{1:t-1} \) is generated according to \( \Theta_p \), the mean of the state estimator \( \hat{x}_{t|t-1}^{(p)} \) is

\[ \alpha_{t}^{(p)} = \mathbb{E}_{p}[\hat{x}_{t|t-1}^{(p)}] = \mu_{t}^{(p)}, \]  \hspace{1cm} (III.22)

and the variance of the estimator \( \hat{x}_{t|t-1}^{(p)} \) is

\[ \phi_{t} = \mathbb{E}_{p}[\hat{x}_{t|t-1}^{(p)}(\hat{x}_{t|t-1}^{(p)})^T] = S_{t}^{(p)} - \hat{V}_{t|t-1}^{(p)}, \]  \hspace{1cm} (III.23)

where \( \hat{x}_{t|t-1}^{(p)} = \hat{x}_{t|t-1}^{(p)} - \mathbb{E}_{p}[\hat{x}_{t|t-1}^{(p)}] \) is the estimator bias.

**Lemma 2** (Mismatched Filter). Suppose the observation \( y_{1:t-1} \) is generated according to \( \Theta_p \), the mean of the state estimator \( \hat{x}_{t|t-1}^{(q)} \) is

\[ \alpha_{t}^{(q)} = \mathbb{E}_{p}[\hat{x}_{t|t-1}^{(q)}] = F_{t-1}^{(q)}\alpha_{t-1}^{(q)} + A_{q}K_{t-1}^{(q)}(\gamma_{t-1}^{(p)} - \bar{y}_{q}), \]  \hspace{1cm} (III.24)
with initial condition \( \alpha_1^{(q)} = \mu_q \). The variance of the estimator is

\[
\xi_t = \mathbb{E}_p [\tilde{x}_{t|t-1}^T (\tilde{x}_{t|t-1})^T] = F_t^{(q)} \xi_{t-1} + F_t^{(q)} \psi_{t-1}^T (A_q K_{t-1}^{(q)} C_p)^T + A_q K_{t-1}^{(q)} C_p \psi_{t-1} (F_t^{(q)})^T + A_q K_{t-1}^{(q)} \Sigma_{t-1}^{(p)} (A_q K_{t-1}^{(q)})^T,
\]

where \( \tilde{x}_{t|t-1}^{(q)} = \tilde{x}_{t|t-1} - \mathbb{E}_p [\tilde{x}_{t|t-1}^{(q)}] \) is the estimator bias, and \( \psi_t \) is the cross-covariance between \( \tilde{x}_{t|t-1}^{(p)} \) and \( \tilde{x}_{t|t-1}^{(q)} \), and

\[
\psi_t = \mathbb{E}_p [\tilde{x}_{t|t-1}^{(p)} (\tilde{x}_{t|t-1}^{(q)})^T] = A_p \psi_{t-1} (F_t^{(q)})^T + A_p \Sigma_{t-1}^{(p)} (A_q K_{t-1}^{(q)} C_p)^T.
\]

The initial conditions are \( \psi_1 = 0 \) and \( \xi_1 = 0 \).

### KL divergence results

We now state the KL divergence results (see Appendix III.H.1 for proofs).

**Theorem 3 (KL divergence rate in image space).** The KL divergence between two DT, \( \Theta_p \) and \( \Theta_q \), in the observation (image) space is

\[
D_r(p(y_{1:t}) \| q(y_{1:t})) = \frac{1}{\tau} \sum_{t=1}^{\tau} D(p(y_t | y_{1:t-1}) \| q(y_t | y_{1:t-1}))
\]

where the conditional KL divergence is

\[
D(p(y_t | y_{1:t-1}) \| q(y_t | y_{1:t-1})) = \frac{1}{2} \log \left| \frac{\hat{S}_{t|t-1}^{(q)}}{\hat{S}_{t|t-1}^{(p)}} \right| + \frac{1}{2} \text{tr} \left( \left( \hat{S}_{t|t-1}^{(q)} \right)^{-1} \Sigma_t^{(p)} \right) - \text{tr} \left( C_q^T \left( \hat{S}_{t|t-1}^{(q)} \right)^{-1} C_p \psi_t \right)
\]

\[
+ \frac{1}{2} \text{tr} \left[ C_q^T (\hat{S}_{t|t-1}^{(q)})^{-1} C_q \psi_t \right] + \frac{1}{2} \left\| C_p \mu_t^{(p)} - C_q \alpha_t^{(q)} + \bar{y}_p - \bar{y}_q \right\|^2_{\hat{S}_{t|t-1}^{(q)}} - \frac{m}{2},
\]

with \{\psi_t, \xi_t, \alpha_t^{(q)}\} computed according to Lemma 2.

**Theorem 4 (Asymptotic KL divergence rate in image space).** Suppose two DT, \( \Theta_p \) and \( \Theta_q \), are both stable (i.e. the eigenvalues of \( A_p \) and \( A_q \) are within the unit
circle), then the asymptotic KL divergence rate between the two DT is

$$D(p(Y) \| q(Y)) = \frac{1}{2} \log \frac{\bar{\Sigma}(q)}{\bar{\Sigma}(p)} + \frac{1}{2} \text{tr} \left[ (\bar{\Sigma}(q))^{-1} \Sigma(p) \right] - \text{tr} \left[ C_q^T (\bar{\Sigma}(q))^{-1} C_p \psi \right]$$

$$+ \frac{1}{2} \text{tr} \left[ C_q^T (\bar{\Sigma}(q))^{-1} C_q \xi \right] + \frac{1}{2} \| \eta \|_{\bar{\Sigma}(q)}^2 - \frac{m}{2},$$  \hspace{1cm} (III.30)

where

$$\eta = (\bar{y}_p - \bar{y}_q) - C_q (I - F_q) A_q K_q (\bar{y}_p - \bar{y}_q),$$  \hspace{1cm} (III.31)

$$\Sigma(p) = C_p S(p) C_p^T + R_p,$$  \hspace{1cm} (III.32)

and $S^{(p)}, \psi, \xi$ satisfy the Sylvester equations

$$S^{(p)} = A_p S^{(p)} A_p^T + Q_p,$$  \hspace{1cm} (III.33)

$$\psi = A_p \psi F_q^T + A_p S^{(p)} (A_q K_q C_p) T,$$  \hspace{1cm} (III.34)

$$\xi = F_q \xi F_q^T + F_q \psi^T (A_q K_q C_p) T + A_q K_q C_p \psi F_q^T$$

$$+ A_q K_q \Sigma^{(p)} (A_q K_q)^T,$$  \hspace{1cm} (III.35)

and

$$F_p = \lim_{t \to \infty} F_t^{(p)}, \quad K_p = \lim_{t \to \infty} K_t^{(p)}, \quad \bar{\Sigma}^{(p)} = \lim_{t \to \infty} \hat{\Sigma}_t^{(p)},$$

$$F_q = \lim_{t \to \infty} F_t^{(q)}, \quad K_q = \lim_{t \to \infty} K_t^{(q)}, \quad \bar{\Sigma}^{(q)} = \lim_{t \to \infty} \hat{\Sigma}_t^{(q)},$$  \hspace{1cm} (III.36)

are the parameters of the steady-state Kalman filters for $\Theta_p$ and $\Theta_q$.

If the observation noise covariances $R_p$ or $R_q$ are full covariance matrices, then computing the log-determinants and matrix inverses in (III.29), (III.30), (III.18) will be computationally intensive if $m$ is large. However, if $R_p$ and $R_q$ are diagonal (or iid) covariance matrices, then the log-determinant can be computed efficiently via simultaneous diagonalization (see Appendix III.H.2), and the matrices inverted with the matrix inversion lemma.

III.C.4 Example

We demonstrate using the KL divergence to compute distances between video patches. Figure III.1 (left) shows a video containing two video textures,
steam surrounded by water. Four $15 \times 15 \times 140$ video patches were extracted, and a DT was learned for each patch using the least-squares method with $n = 5$ (the observation noise was assumed i.i.d.). Figure III.1 (right) plots the KL divergence rate between patches \{“B”, “C”, “D”\} and patch “A”, for values of $\tau \in \{1, 2, 3, \ldots, 10000\}$. In this example, the divergence converges to the asymptotic KL divergence rate (plotted as a dashed line) around $\tau > 1000$.

### III.D Martin distance

The Martin distance [45], introduced in [16] as a distance metric between two dynamic textures, is based on the principal angles between the subspaces of the extended observability matrices of the two textures [46]. Formally, let $\Theta_p$ and $\Theta_q$ be the parameters representing two dynamic textures, then the Martin distance is defined as

$$d(\Theta_p, \Theta_q)^2 = -\log \prod_{i=1}^n \cos^2 \theta_i,$$

(III.37)

where $\theta_i$ is the i-th principal angle between the extended observability matrices $\mathcal{O}_p$ and $\mathcal{O}_q$, defined as

$$\mathcal{O}_i = \left[ C_i^T \quad A_i^T C_i^T \quad \ldots \quad (A_i^T)^n C_i^T \quad \ldots \right]^T,$$  

for $i \in \{p, q\}$. It
is shown in [46] that the principal angles can be computed as the solution to the following generalized eigenvalue problem:

$$
\begin{bmatrix}
0 & O_{pq} \\
(O_{pq})^T & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix} = \lambda
\begin{bmatrix}
O_{pp} & 0 \\
0 & O_{qq}
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
$$  \hspace{1cm} (III.38)

subject to $x^T O_{pp} x = 1$ and $y^T O_{qq} y = 1$, where

$$
O_{ij} = (O_i)^T O_j = \sum_{t=0}^{\infty} (A_i^t)^T C_i^T C_j A_j^t
$$  \hspace{1cm} (III.39)

for $i \in \{p, q\}$ and $j \in \{p, q\}$. The first $n$ largest eigenvalues are the cosines of the principal angles, i.e. $\lambda_i = \cos \theta_i$ for $i = 1, 2, \ldots, n$, and hence

$$
d(\Theta_p, \Theta_q)^2 = -2 \sum_{i=1}^{n} \log \lambda_i.
$$  \hspace{1cm} (III.40)

The infinite sum of (III.39) can obtained by solving the corresponding Sylvester equation

$$
O_{ij} = A_i^T O_{ij} A_j + C_i^T C_j,
$$  \hspace{1cm} (III.41)

which can be computed algorithmically in $O(n^3)$ time. Note that the Martin distance does not exploit the probabilistic framework of the dynamic texture, as the distance is only computed using the $A$ and $C$ parameters. This may hinder the effectiveness of the Martin distance when stochastic nature of the texture is discriminant.

### III.E Applications to traffic monitoring and video texture classification

In recent years, the use of video systems for traffic monitoring has shown promise over that of traditional loop detectors. The analysis of traffic video can provide global information, such as overall traffic speed, lane occupancy, and individual lane speed, along with the capability to track individual cars. Because
video systems are less disruptive and less costly to install than loop detectors, interest has grown in building and using large camera networks to monitor different aspects of traffic, such as traffic congestion.

Most of the existing work in monitoring traffic uses a vehicle segmentation and tracking framework. First, a potential vehicle is segmented from the scene using motion cues [47] [48], or through background subtraction [49]. Once segmentation is performed, the objects are tracked between frames using rule-based systems [47], Kalman filters, or Condensation [9]. In [49], object trajectories are represented as polynomial curves, which are used for video retrieval. The vehicle tracking framework has the disadvantage that its accuracy is dependent on the quality of the segmentation. The segmentation task becomes more difficult with the presence of adverse environmental conditions, such as lighting (e.g. overcast, glare, night), shadows, occlusion, and blurring. Furthermore, segmentation cannot be performed reliably on low resolution images where the vehicles only span a few pixels. Tracking algorithms also have problems when there are many objects in the scene, which is typically the case for highways scenes with congestion.

Several recent methods finesse the problems associated with vehicle tracking by analyzing the low-level motion vectors provided by MPEG video compression. In [50], the MPEG motion vector field is filtered to remove vectors that are not consistent with vehicle motion, and traffic flow is estimated by averaging the remaining motion vectors. The work of [51] uses a probabilistic approach that models each category of traffic as a Gaussian-mixture hidden Markov model (GM-HMM), which is learned from feature vectors extracted from the MPEG video stream. Classification is performed by selecting the category corresponding to the GM-HMM of largest likelihood for the query video. While these two methods do not rely on vehicle tracking, they depend on the estimation of motion vectors, which may be subject to noise when there are many vehicles in the scene.

Since most of the information required for the classification of traffic video is contained in the interaction between the many motions that it contains, a holistic
representation can be used to capture the variability of the motion field without
the need for segmenting or tracking individual components. Hence, we propose to
model the entire motion field as a dynamic texture. Using the KL divergence or
Martin distance, traffic videos similar to a query can be retrieved, or the traffic
congestion of the query can be classified using a nearest neighbors classifier or SVM.
Since only the motion is modeled, the proposed framework is inherently invariant
to lighting changes. In addition, because the model does not rely on a dense
motion field based on pixel similarity (e.g. correlation or optical flow), it is robust
to occlusion, blurring, image resolution, and other image transformations. In the
remainder of this section, we evaluate the performance of motion flow recognition
using the KL kernel and dynamic textures in the context of traffic monitoring. We
also present results on video texture classification. Video results are available in
the supplemental [52].

III.E.1 Databases

We evaluate the performance of motion flow recognition using the KL
kernel on two video databases. The first database, based on traffic video, contains
visually similar classes, but with varying temporal characteristics. The second
database contains many visually distinct classes.

| Figure III.2 Frames from a video of highway traffic. (courtesy Washington State Department of Transportation) |
Traffic video database

The traffic video database consists of 254 video sequences of highway traffic in Seattle, collected from a single stationary traffic camera over two days [53]. The database contains a variety of traffic patterns and weather conditions (e.g. overcast, raining, sunny, rain drops on the camera lens), and an example video is shown in Figure III.2. Each video was recorded in color with a resolution of 320 x 240 pixels with between 42 to 52 frames at 10 fps. Each sequence was converted to grayscale, resized to 80 x 60 pixels, and then clipped to a 48 x 48 window over the area with the most total motion. Finally, for each video clip, the mean image was subtracted and the pixel intensities were normalized to have unit variance. This was done to reduce the impact of the different lighting conditions.

The database was labeled by hand with respect to the amount of traffic congestion in each sequence. In total there were 44 sequences of heavy traffic (slow or stop and go speeds), 45 of medium traffic (reduced speed), and 165 of light traffic (normal speed). Figure III.3 (left) shows a representative set of clips from this database. All clips are very similar in that the views are obtained with a fixed camera facing the same stretch of road, and the motion is always in the same direction and confined to the same area. Thus, an effective classifier for this problem must be able to distinguish between the different patterns of flow, i.e. the underlying temporal process.
Video texture database

The video texture database used in [16] contains 50 classes of various texture, including boiling water, fountains, fire, rippling water, waterfalls, and plants and flowers swaying in the wind. Each class contains four grayscale video sequences with 75 frames of $160 \times 110$ pixels. Each sequence was clipped to a $48 \times 48$ window that contained the representative motion. Figure III.3 (right) shows several examples of the video patches from the dynamic texture database. Since almost all of the classes are visually distinct, the appearance component of the model is likely to be as important for classification as the motion component.

III.E.2 Experiment setup

The parameters of the dynamic texture model were learned for each video clip using the least-squares method of Section II.D.2. To ensure that the KL divergence converges, the transition matrix $A$ was scaled so that the largest eigenvalues lie inside the unit circle. In addition, the covariance of the driving process was regularized to prevent problems with singular matrices, i.e. we set $Q' = Q + I_n$. All classification results were averaged over four trials. In each trial the data set was split differently with 75% used for training and cross-validation, and 25% reserved for testing.

For the traffic video database, the SVMs were trained with the KL kernel between state spaces ($\tau = 250$), and for the video texture database, SVMs were trained using the KL kernel in the image space ($\tau = 25$). A one-versus-all scheme was used to learn the multi-class problem, and the $C$ and $\gamma$ parameters of the SVM were selected using 3-fold cross-validation over the training set. The SVM training and testing was performed using the libsvm software package [54]. We also tested a nearest neighbor (NN) classifier using the image space and state space KL as distance measures. Finally, for comparison with the state-of-the-art, a nearest neighbor classifier was implemented using the Martin distance [45, 46] as

---

1The four videos in each class originate from 2 videos with 150 frames each.
suggested in [16]. For this experiment, the extended observability matrices were approximated with $\tau = 250$.

### III.E.3 Retrieval results for traffic video

The motion model was tested in a video retrieval experiment where the goal was to retrieve instances of traffic patterns that were similar to the query video. For the experiment, dynamic texture models with $n = 15$ principal components were used, and the state KL divergence and Martin distance were used as the similarity measures. The precision-recall curve for traffic video retrieval is presented in Figure III.4. The Martin distance and state KL divergence perform similarly, with the state KL divergence performing slightly better. Figure III.5 shows the results of several queries for light, medium, and heavy traffic using the state KL divergence. A query using nighttime sequences outside of the original database is also presented in the lower half of Figure III.5, and shows that the retrieval system is robust to variable lighting conditions. In addition the framework is robust to occlusion and blurring due to raindrops on the camera lens, as seen in the 3rd and 5th results of Figure III.5e.
Figure III.5 Video retrieval results for (a) light traffic, (b) medium traffic, and (c) heavy traffic during the day. Retrieval using a night sequence outside the original database for (d) light, (e) medium, and (f) heavy traffic shows robustness to lighting conditions.
III.E.4 Classification results on traffic video

Figure III.6 (left) presents the classification results obtained on the traffic video database. It can be seen from this figure that the two state KL classifiers outperform the Martin NN classifier on this database. Furthermore, all classifiers improve as the number of principal components increases, confirming the fact that a static classifier would do rather poorly on this database. Comparing the performance of the state KL classifiers versus the Martin NN counterpart it can be concluded that 1) the SVM-KL combination is consistently better, and 2) the NN-KL combination is better for $n \geq 15$ and also achieves a higher maximum accuracy. The best classifier, the state KL-SVM with 15 components, has an overall accuracy of 94.5%. Table III.1 shows the confusion matrix for the best classifier, averaged over the four test trials.

Figure III.7 shows several classification examples under different lighting conditions: (a) sunny lighting, including strong shadows; and (b) overcast lighting, including raindrops on the camera lens. Several night time videos outside the original database were also fed through the same classifier. Even though the classifier was trained with video taken during the day, it is still able to correctly
Table III.1 Confusion matrix for traffic congestion classification

<table>
<thead>
<tr>
<th>PREDICTED</th>
<th>heavy</th>
<th>medium</th>
<th>light</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heavy</td>
<td>37</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>medium</td>
<td>4</td>
<td>39</td>
<td>2</td>
</tr>
<tr>
<td>light</td>
<td>0</td>
<td>1</td>
<td>164</td>
</tr>
</tbody>
</table>

label the nighttime video sequences, including the event of a traffic jam (heavy traffic) at night. This is particularly interesting since the classifiers were trained with daytime images containing normally lit cars, yet they are able to correctly label nighttime images where the cars are represented as headlights and a pair of tail lights. These results provide evidence that the dynamic texture model is indeed extracting relevant motion information, and that the proposed classification framework is capable of using the motion model to discriminate between classes of motion.

A final experiment was conducted to identify the traffic pattern of the highway during the day. The SVM and nearest neighbor classifiers were trained using 61 sequences spanning 4 hours from the first day, and tested on 193 sequences
spanning 15 hours from the following day. The ground truth classification and the outputs of the state-KL SVM, state-KL NN, and Martin distance NN classifiers are shown in Figure III.8. The increase in traffic due to rush hour can be seen between 2:30 PM and 6:30 PM.

Figure III.8 Classification of congestion in traffic sequences spanning 15 hours: (top to bottom) ground truth; classification using state KL SVM, state KL nearest neighbors, and Martin distance nearest neighbors. Errors are highlighted with circles.

III.E.5 Classification results on video textures

The results on the dynamic texture database, presented in Figure III.6 (right), show that the image-based KL classifiers performs significantly better than the Martin distance classifier (an improvement of the best classification accuracy from 89% to 96%). Note that the accuracies of the image KL classifiers improve as the number of principal components \( n \) decreases. This was expected since the dynamic texture database contains many visually distinct classes for which the appearance components is more discriminant than the motion. In fact, in the degenerate case of \( n = 0 \), the video is modeled as a Gaussian whose mean is the mean image of the video sequence, and covariance is the deviation of the frames from this mean. Note how, by achieving top performance for a small number of components, the image-based KL classifiers virtually become static classifiers.
In contrast, the Martin distance nearest neighbors classifier does rather poorly with a small number of components. Hence, although performance improves as \( n \) increases, it never reaches an accuracy comparable to that of the KL-based classifiers.

**III.F Summary and discussion**

In this chapter, we have presented a framework for classifying motion that combines the the modeling power of the dynamic texture and the generalization guarantees of the SVM. This combination is achieved by the derivation of a new probabilistic kernel based on the KL divergence between dynamic textures.

Overall, the image and state KL classifiers outperform the Martin distance nearest neighbor method in classification tasks with both visually distinct video textures, and visually similar, but temporally distinct, video textures. The KL classifiers are also capable of spanning the gamut from static to highly-varying dynamic classifier and, therefore, provide a generic framework for the classification of a large variety of video streams. Comparing the performance of the two KL classifiers, it is clear that SVM-KL combination achieves better classification performance than NN-KL. In particular, the greater robustness of the SVM classifier to a poor selection of the number of components indicates that it has better generalization ability.

We have also presented a method for modeling traffic flow patterns holistically using dynamic texture models. When compared to previous solutions, the analysis of motion with these models has several advantages: a) it does not require segmentation or tracking of vehicles; b) it does not require estimation of a dense motion vector field; and c) it is robust to lighting variation, blurring, occlusion, and low image resolution. Experimental results using the holistic model show good performance in the domains of video retrieval and classification of traffic congestion. While the system proposed in this work was trained and tested on a single camera
view, it can be augmented to handle multiple camera views by transforming the
video from each camera view into a common frame of reference.

III.G Acknowledgements

We wish to thank Gianfranco Doretto and Stefano Soatto for providing
the video texture database used in [16], and the Washington State Department of
Transportation for the traffic video.

The text of Chapter III, in full, is based on the material as it appears
in: A. B. Chan and N. Vasconcelos, “Probabilistic kernels for the classification of
auto-regressive visual processes”, in IEEE Conf. on Computer Vision and Pattern
Recognition (CVPR), vol. 1, pp. 846–851, 2005; A. B. Chan and N. Vasconcelos,
“On the KL divergence between linear dynamical systems”, in preparation;
A. B. Chan and N. Vasconcelos, “Classification and retrieval of traffic video using
auto-regressive stochastic processes”, in IEEE Intelligent Vehicles Symposium
(IEEEIV), pp. 771–776, 2005. The dissertation author was a primary researcher
and an author of the cited material.

III.H Appendix

III.H.1 Proofs for KL divergence in image space

In this appendix, we first derive some useful properties, followed by the
proofs for the sensitivity analysis of the Kalman filter, and for the KL divergence
between DT. We will use the same notation as in Section III.C.3.

Useful properties

We have the following properties:

Property 1. The covariance between $\hat{x}_{t|t-1}^{(q)}$ and $y_t$ is

$$
E_p \left[ \hat{x}_{t|t-1}^{(q)} (y_t - E_p( y_t ))^T \right] = \psi_t^T C_p^T, \tag{III.42}
$$
and the covariance between \( \hat{x}_{t|t-1}^{(p)} \) and \( y_t \) is

\[
\mathbb{E}_p \left[ \hat{x}_{t|t-1}^{(p)} (y_t - \mathbb{E}_p(y_t))^T \right] = \phi_t C_p^T. \tag{III.43}
\]

**Proof.**

\[
\begin{align*}
\mathbb{E}_p \left[ \hat{x}_{t|t-1}^{(q)} (y_t - \mathbb{E}_p(y_t))^T \right] \\
= \int p(y_{1:t}) \hat{x}_{t|t-1}^{(q)} (y_t - \mathbb{E}_p(y_t))^T \\
= \int p(y_{1:t-1}) p(y_{1:t-1}|y_{1:t-1}) \hat{x}_{t|t-1}^{(q)} (y_t - \mathbb{E}_p(y_t))^T \\
= \int p(y_{1:t-1}) \hat{x}_{t|t-1}^{(q)} \int p(y_{1:t-1}^p|y_{1:t-1}^p) (y_t - \mathbb{E}_p(y_{1:t-1}^p))^T dy_t dy_{1:t-1} \\
= \mathbb{E}_p \left[ \hat{x}_{t|t-1}^{(q)} (\hat{y}_{t|t-1}^{(p)} - \mathbb{E}_p(\hat{y}_{t|t-1}^{(p)}))^T \right] \\
= \mathbb{E}_p \left[ \hat{x}_{t|t-1}^{(q)} (C_p \hat{x}_{t|t-1}^{(p)} + \bar{y}_p - C_p \mathbb{E}_p(\hat{x}_{t|t-1}^{(p)}) - \bar{y}_p)^T \right] \\
= \mathbb{E}_p \left[ \hat{x}_{t|t-1}^{(q)} (C_p \hat{x}_{t|t-1}^{(p)})^T \right] \\
= \psi_t^T C_p^T, \tag{III.50}
\end{align*}
\]

where in (III.46) we have used the fact that

\[
\begin{align*}
\mathbb{E}_p[\hat{y}_{t|t-1}^{(p)}] &= \int p(y_{1:t-1}) \left( \int p(y_t|y_{1:t-1}) y_t dy_t \right) dy_{1:t-1} \tag{III.51} \\
&= \int p(y_{1:t}) y_t dy_{1:t} \tag{III.52} \\
&= \mathbb{E}_p[y_t]. \tag{III.53}
\end{align*}
\]

A similar derivation yields (III.43).

**Property 2.** Estimator biases \( \hat{x}_{t|t-1}^{(p)} \) and \( \hat{x}_{t|t-1}^{(q)} \) can be expressed recursively

\[
\begin{align*}
\hat{x}_{t|t-1}^{(p)} &= F_{t-1}^{(p)} \hat{x}_{t-1|t-1}^{(p)} + A_p K_{t-1}^{(p)} (y_t - \mathbb{E}_p[y_{t-1}]), \tag{III.54} \\
\hat{x}_{t|t-1}^{(q)} &= F_{t-1}^{(q)} \hat{x}_{t-1|t-1}^{(q)} + A_q K_{t-1}^{(q)} (y_t - \mathbb{E}_p[y_{t-1}]). \tag{III.55}
\end{align*}
\]


Proof.

\[
\hat{x}_{t|t-1}^{(p)} = \hat{x}_{t|t-1}^{(p)} - \mathbb{E}_p[\hat{x}_{t|t-1}^{(p)}] 
\]  (III.56)

\[
= F_{t-1}^{(p)} \hat{x}_{t-1|t-2}^{(p)} + A_p K_{t-1}^{(p)} (y_{t-1} - \bar{y}_p) - \mathbb{E}_p[\hat{x}_{t-1|t-2}^{(p)}] 
\]  (III.57)

\[
+ A_p K_{t-1}^{(p)} (y_{t-1} - \bar{y}_p) 
\]  (III.58)

\[
= F_{t-1}^{(p)} \hat{x}_{t-1|t-2}^{(p)} + A_p K_{t-1}^{(p)} (y_{t-1} - \mathbb{E}_p[y_{t-1}]). 
\]  (III.59)

A similar derivation yields (III.55).

Also, note that the conditional covariances \( \hat{V}_{t|t-1} \) and \( \hat{\Sigma}_{t|t-1} \) are not a function of the observed sequence \( y_{1:t-1} \) when computed with (III.20), and hence the conditional covariances are unchanged when taking expectations over \( y_{1:t-1} \), i.e.

\[
\mathbb{E}[\hat{V}_{t|t-1}] = \int p(y_{1:t-1}) \hat{V}_{t|t-1} dy_{1:t-1} = \hat{V}_{t|t-1}, \quad (III.60) 
\]

\[
\mathbb{E}[\hat{\Sigma}_{t|t-1}] = \int p(y_{1:t-1}) \hat{\Sigma}_{t|t-1} dy_{1:t-1} = \hat{\Sigma}_{t|t-1}. \quad (III.61) 
\]

Sensitivity analysis proofs

Proof for Lemma 1 (Matched filter). The mean of the matched filter \( \hat{x}_{t|t-1}^{(p)} \) is

\[
\alpha_t^{(p)} = \mathbb{E}_p[\hat{x}_{t|t-1}^{(p)}] 
\]  (III.62)

\[
= \mathbb{E}_p[\mathbb{E}_p(x_t|y_{1:t-1})] 
\]  (III.63)

\[
= \mathbb{E}_p[x_t] = \mu_t^{(p)}. \quad (III.64) 
\]
The variance of the matched filter $\hat{x}_{t|t-1}^{(p)}$ is

$$\phi_t = \mathbb{E}_p[\hat{x}_{t|t-1}^{(p)} (\hat{x}_{t|t-1}^{(p)})^T]$$ (III.65)

$$= \mathbb{E}_p[\hat{x}_{t|t-1}^{(p)} (\hat{x}_{t|t-1}^{(p)})^T] - \alpha_t^{(p)} (\alpha_t^{(p)})^T$$ (III.66)

$$= \mathbb{E}_p[\mathbb{E}_p(x_t|y_{1:t-1}) \mathbb{E}_p(x_t|y_{1:t-1})^T] - \mathbb{E}_p[x_t] \mathbb{E}_p[x_t]^T$$ (III.67)

$$= \mathbb{E}_p[\mathbb{E}_p(x_t x_t^T|y_{1:t-1}) - \text{cov}_p(x_t|y_{1:t-1})] - \mathbb{E}_p[x_t] \mathbb{E}_p[x_t]^T$$ (III.68)

$$= \mathbb{E}_p[\mathbb{E}_p(x_t x_t^T|y_{1:t-1}) - \hat{V}_{t|t-1}^{(p)}] - \mathbb{E}_p[x_t] \mathbb{E}_p[x_t]^T - \hat{V}_{t|t-1}^{(p)}$$ (III.69)

$$= \mathbb{E}_p[x_t x_t^T] - \mathbb{E}_p[x_t] \mathbb{E}_p[x_t]^T - \hat{V}_{t|t-1}^{(p)}$$ (III.70)

$$= \text{cov}_p(x_t) - \hat{V}_{t|t-1}^{(p)} = S_t^{(p)} - \hat{V}_{t|t-1}^{(p)},$$ (III.71)

where (III.68) follows from the definition of covariance, i.e. $\text{cov}(x_t) = \mathbb{E}[x_t x_t^T] - \mathbb{E}[x_t] \mathbb{E}[x_t]^T$, and (III.70) follows from (III.60).

\[ \square \]

**Proof for Lemma 2 (Mismatched filter).** The mean of the mismatched estimator $\hat{x}_{t|t-1}^{(q)}$ is obtained by substituting (III.21)

$$\alpha_t^{(q)} = \mathbb{E}_p[\hat{x}_{t|t-1}^{(q)}] = \mathbb{E}_p \left[ F_t^{(q)} \hat{x}_{t-1|t-2}^{(q)} + A_q K_t^{(q)} (y_{t-1} - \bar{y}_q) \right]$$ (III.73)

$$F_t^{(q)} \alpha_t^{(q)} + A_q K_t^{(q)} (\mathbb{E}_p[y_{t-1}] - \bar{y}_q),$$ (III.74)

with initial condition $\alpha_1^{(q)} = \mathbb{E}_p[\hat{x}_{1|0}^{(q)}] = \mu_q$.

The cross-covariance between the estimators $\hat{x}_{t|t-1}^{(p)}$ and $\hat{x}_{t|t-1}^{(q)}$ can be written recursively by substituting (III.54) and (III.55),

$$\psi_t = \mathbb{E}_p[\hat{x}_{t|t-1}^{(p)} (\hat{x}_{t|t-1}^{(q)})^T]$$ (III.76)

$$= \mathbb{E}_p \left[ (F_t^{(p)} \hat{x}_{t-1|t-2}^{(p)} + A_p K_t^{(p)} (y_{t-1} - \mathbb{E}_p[y_{t-1}])) \right] \left( (F_t^{(q)} \hat{x}_{t-1|t-2}^{(q)} + A_q K_t^{(q)} (y_{t-1} - \mathbb{E}_p[y_{t-1}]))^T \right).$$ (III.77)
Expanding the product and taking the expectation using (III.42) and (III.43) yields

\[ \psi_t = F_{t-1}^{(p)} \psi_{t-1} (F_{t-1}^{(q)})^T + F_{t-1}^{(p)} \phi_{t-1} C_p^T (A_q K_{t-1}^{(q)})^T \]

\[ + A_p K_{t-1}^{(p)} C_p \psi_{t-1} (F_{t-1}^{(q)})^T + A_p K_{t-1}^{(p)} \text{cov}_p (y_{t-1}) (A_q K_{t-1}^{(q)})^T \]  

(III.78)

\[ = A_p \psi_{t-1} (F_{t-1}^{(q)})^T + (F_{t-1}^{(p)} \phi_{t-1} C_p^T + A_p K_{t-1}^{(p)} \text{cov}_p (y_{t-1})) (A_q K_{t-1}^{(q)})^T. \]  

(III.79)

The underlined term can be simplified

\[ F_{t-1}^{(p)} \phi_{t-1} C_p^T + A_p K_{t-1}^{(p)} \text{cov}_p (y_{t-1}) \]

\[ = (A_p - A_p K_{t-1}^{(p)} C_p) \phi_{t-1} C_p^T + A_p K_{t-1}^{(p)} \text{cov}_p (y_{t-1}) \]  

(III.80)

\[ = A_p \phi_{t-1} C_p^T + A_p K_{t-1}^{(p)} (\text{cov}_p (y_{t-1}) - C_p \phi_{t-1} C_p^T) \]  

(III.81)

\[ = A_p \phi_{t-1} C_p^T + A_p K_{t-1}^{(p)} (C_p (\text{cov}_p (x_{t-1}) - \phi_{t-1}) C_p^T + R) \]  

(III.82)

\[ = A_p \phi_{t-1} C_p^T + A_p K_{t-1}^{(p)} (C_p \hat{\phi}_p (t_{t-1}) C_p^T + R) \]  

(III.83)

\[ = A_p (\text{cov}_p (x_{t-1}) - \hat{V}_p (t_{t-1}) C_p^T + A_p K_{t-1}^{(p)} \hat{\phi}_p (t_{t-1} C_p) + R) \]  

(III.84)

\[ = A_p (\text{cov}_p (x_{t-1}) - \hat{V}_p (t_{t-1}) C_p^T + A_p \hat{V}_p (t_{t-1}) C_p) \]  

(III.85)

\[ = A_p \text{cov}_p (x_{t-1}) C_p^T, \]  

(III.86)

where (III.83) follows from (III.72). Finally, substituting (III.86) into (III.79) yields

\[ \psi_t = A_p \psi_{t-1} (F_{t-1}^{(q)})^T + A_p \text{cov}_p (x_{t-1}) (A_q K_{t-1}^{(q)} C_p)^T \]  

(III.87)

with initial condition \( \psi_1 = \mathbb{E}_p [(\hat{x}_{1|0}^{(p)} - \mu_p)(\hat{x}_{1|0}^{(q)} - \mu_q)^T] = 0 \).

The variance of the mismatched filter \( \hat{x}_{t|t-1}^{(q)} \) can be written recursively by substituting (III.55)

\[ \xi_t = \mathbb{E}_p [\hat{x}_{t|t-1}^{(q)} (\hat{x}_{t|t-1}^{(q)})^T] \]  

(III.88)

\[ = \mathbb{E}_p \left[ (F_{t-1}^{(q)} \hat{x}_{t-1|t-2}^{(q)} A_q K_{t-1}^{(q)} (y_{t-1} - \mathbb{E}_p [y_{t-1}])) . \right. \]

(III.89)

\[ \left. (F_{t-1}^{(q)} \hat{x}_{t-1|t-2}^{(q)} A_q K_{t-1}^{(q)} (y_{t-1} - \mathbb{E}_p [y_{t-1}]))^T \right]. \]

Expanding the product and taking the expectation using (III.42) yields

\[ \xi_t = F_{t-1}^{(q)} \xi_{t-1} (F_{t-1}^{(q)})^T + F_{t-1}^{(q)} \psi_{t-1} (A_q K_{t-1}^{(q)} C_p)^T \]

\[ + A_q K_{t-1}^{(q)} C_p \psi_{t-1} (F_{t-1}^{(q)})^T + A_q K_{t-1}^{(q)} \text{cov}_p (y_{t-1}) (A_q K_{t-1}^{(q)})^T \]  

(III.90)
with initial condition \( \xi_1 = \mathbb{E}_p[(\hat{x}^{(q)}_{1|0} - \mu_q)(\hat{x}^{(q)}_{1|0} - \mu_q)^T] = 0. \)

KL divergence proofs

Proof for Theorem 3 (KL divergence rate in image space). Using the chain rule for KL divergence [40], (III.4) can be rewritten in terms of a sum of conditional KL divergences

\[
D_r(p(y_{1:t}) \| q(y_{1:t})) = \frac{1}{\tau} \sum_{t=1}^{\tau} D(p(y_t|y_{1:t-1}) \| q(y_t|y_{1:t-1})),
\]

(III.91)

where the conditional KL divergence is defined as

\[
D(p(y_t|y_{1:t-1}) \| q(y_t|y_{1:t-1})) = \int p(y_{1:t-1}) \int p(y_t|y_{1:t-1}) \log \frac{p(y_t|y_{1:t-1})}{q(y_t|y_{1:t-1})} dy_{1:t}
\]

\[= \mathbb{E}_p \left( \int p(y_t|y_{1:t-1}) \log \frac{p(y_t|y_{1:t-1})}{q(y_t|y_{1:t-1})} dy_t \right),
\]

(III.92)

and \( \mathbb{E}_p \) is the expectation with respect to \( p(y_{1:t-1}) \). Distributions \( p(y_t|y_{1:t-1}) \) and \( q(y_t|y_{1:t-1}) \) are conditional Gaussian distributions with means \( \hat{y}^{(p)}_{t|t-1} \) and \( \hat{y}^{(q)}_{t|t-1} \) and covariances \( \hat{\Sigma}^{(p)}_{t|t-1} \) and \( \hat{\Sigma}^{(q)}_{t|t-1} \), hence the integral in (III.92) is the KL divergence between two Gaussians [40]

\[
D(p(y_t|y_{1:t-1}) \| q(y_t|y_{1:t-1}))
\]

(III.93)

\[
= \mathbb{E}_p \left[ \frac{1}{2} \log \frac{\hat{\Sigma}^{(q)}_{t|t-1}}{\hat{\Sigma}^{(p)}_{t|t-1}} + \frac{1}{2} \text{tr} \left( \hat{\Sigma}^{(q)}_{t|t-1}^{-1} \hat{\Sigma}^{(p)}_{t|t-1} \right) + \frac{1}{2} \left\| \hat{y}^{(p)}_{t|t-1} - \hat{y}^{(q)}_{t|t-1} \right\|^2_{\hat{\Sigma}^{(q)}_{t|t-1}} - \frac{m}{2} \right],
\]

where \( \|x\|^2_A = x^T A^{-1} x \). The expectations of the first three terms in (III.93) are:

- Using (III.61), the log-determinant term is
  \[
  \mathbb{E}_p \left[ \log \frac{\hat{\Sigma}^{(q)}_{t|t-1}}{\hat{\Sigma}^{(p)}_{t|t-1}} \right] = \log \frac{\hat{\Sigma}^{(q)}_{t|t-1}}{\hat{\Sigma}^{(p)}_{t|t-1}}.
  \]
  (III.94)

- Using (III.61), the trace term is
  \[
  \mathbb{E}_p \left[ \text{tr} \left( \hat{\Sigma}^{(q)}_{t|t-1}^{-1} \hat{\Sigma}^{(p)}_{t|t-1} \right) \right] = \text{tr} \left( \hat{\Sigma}^{(q)}_{t|t-1}^{-1} \hat{\Sigma}^{(p)}_{t|t-1} \right).
  \]
  (III.95)
Again using (III.61), the Mahalanobis distance term is

\[
E_p \left[ \| \hat{y}_t^{(p)} - \hat{y}_t^{(q)} \|^2 \right] = E_p \left[ \text{tr}[(\hat{\Sigma}_t^{(q)})^{-1} (\hat{y}_t^{(p)} - \hat{y}_t^{(q)}) (\hat{y}_t^{(p)} - \hat{y}_t^{(q)})^T] \right]
\]

For convenience, define the following

\[
\eta_t^{(p)} = E_p[\hat{y}_t^{(p)}] = E_p[C_p \hat{x}_{t-1}^{(p)} + \bar{y}_p] = C_p \alpha_t^{(p)} + \bar{y}_p,
\]

\[
\eta_t^{(q)} = E_p[\hat{y}_t^{(q)}] = E_p[C_q \hat{x}_{t-1}^{(q)} + \bar{y}_q] = C_q \alpha_t^{(q)} + \bar{y}_q.
\]

Note that

\[
\hat{y}_t^{(p)} = C_p \hat{x}_{t-1}^{(p)} + \bar{y}_p = C_p \hat{x}_{t-1}^{(p)} - C_p \alpha_t^{(p)} + C_p \alpha_t^{(p)} + \bar{y}_p = C_p \hat{x}_{t-1}^{(p)} + \eta_t^{(p)}
\]

and likewise

\[
\hat{y}_t^{(q)} = C_q \hat{x}_{t-1}^{(q)} + \eta_t^{(q)}.
\]

Substituting (III.103) and (III.104), the expectation becomes

\[
E_p \left[ (\hat{y}_t^{(p)} - \hat{y}_t^{(q)}) (\hat{y}_t^{(p)} - \hat{y}_t^{(q)})^T \right]
\]

\[
= E_p \left[ \left( (C_p \hat{x}_{t-1}^{(p)} - C_q \hat{x}_{t-1}^{(q)}) + \eta_t^{(p)} - \eta_t^{(q)} \right) \left( (C_p \hat{x}_{t-1}^{(p)} - C_q \hat{x}_{t-1}^{(q)}) + \eta_t^{(p)} - \eta_t^{(q)} \right)^T \right]
\]

\[
= E_p \left[ (C_p \hat{x}_{t-1}^{(p)} - C_q \hat{x}_{t-1}^{(q)})(C_p \hat{x}_{t-1}^{(p)} - C_q \hat{x}_{t-1}^{(q)})^T \right] + (\eta_t^{(p)} - \eta_t^{(q)}) (\eta_t^{(p)} - \eta_t^{(q)})^T
\]

\[
= C_p \Phi_p C_p^T - C_p \psi_t C_q^T - C_q \psi_t^T C_p + C_q \xi_t C_q^T + (\eta_t^{(p)} - \eta_t^{(q)}) (\eta_t^{(p)} - \eta_t^{(q)})^T
\]
where (III.107) follows from $\mathbb{E}_p[C_p \hat{\gamma}^{(p)}_{t|t-1} - C_q \hat{\gamma}^{(q)}_{t|t-1}] = 0$. Finally, substituting into (III.98), the Mahalanobis distance term is

$$
\mathbb{E}_p \left[ \left\| \hat{\gamma}^{(p)}_{t|t-1} - \hat{\gamma}^{(q)}_{t|t-1} \right\|_2^2 \right] = \text{tr} \left[ C_p^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} C_p \phi_t \right] - 2\text{tr} \left[ C_q^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} C_p \psi_t \right] + \text{tr} \left[ C_q^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} - \text{tr} \left[ C_q^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} - \text{tr} \right] \left( \hat{\Sigma}^{(q)}_{t|t-1} \right) \right].
$$

(III.109)

Substituting (III.94), (III.95), and (III.109) into (III.93), the conditional KL divergence becomes

$$
D(p(y_t|y_{1:t-1}) \| q(y_t|y_{1:t-1})) = \frac{1}{2} \log \frac{\hat{\Sigma}^{(q)}_{t|t-1}}{\hat{\Sigma}^{(p)}_{t|t-1}} + \frac{1}{2} \text{tr} \left[ C_p^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} C_p \phi_t \right] - 2\text{tr} \left[ C_q^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} C_p \psi_t \right] + \frac{1}{2} \left\| \eta^{(p)}_t - \eta^{(q)}_t \right\|_2^2.
$$

(III.110)

The underlined term can be simplified

$$
\text{tr} \left[ (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} (\hat{\Sigma}^{(p)}_{t|t-1}) \right] + \text{tr} \left[ C_p^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} C_p \phi_t \right] = \text{tr} \left[ (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} (\hat{\Sigma}^{(p)}_{t|t-1} + C_p \phi_t C_p^T) \right] = \text{tr} \left[ (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} (C_p \hat{\Theta}^{(p)}_{t|t-1} C_p^T + R_p + C_p (\text{cov}_p(x_t) - \hat{\Theta}^{(p)}_{t|t-1}) C_p^T) \right] (III.112)
$$

$$
= \text{tr} \left[ (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} (C_p \text{cov}_p(x_t) C_p^T + R_p) \right] (III.113)
$$

$$
= \text{tr} \left[ (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} \text{cov}_p(y_t) \right].
$$

(III.115)

and hence, the conditional KL divergence is

$$
D(p(y_t|y_{1:t-1}) \| q(y_t|y_{1:t-1})) = \frac{1}{2} \log \frac{\hat{\Sigma}^{(q)}_{t|t-1}}{\hat{\Sigma}^{(p)}_{t|t-1}} + \frac{1}{2} \text{tr} \left[ (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} \text{cov}_p(y_t) \right] - \text{tr} \left[ C_q^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} C_q \psi_t \right] + \frac{1}{2} \text{tr} \left[ C_q^T (\hat{\Sigma}^{(q)}_{t|t-1})^{-1} C_q \xi_t \right] + \frac{1}{2} \left\| \eta^{(p)}_t - \eta^{(q)}_t \right\|_2^2 - \frac{m}{2}.
$$

(III.116)
Proof for Theorem 4 (Asymptotic KL divergence rate in image space). The KL divergence rate is equivalent to the limit of the conditional KL divergence [40]

\[
D(P\|Q) = \lim_{\tau \to \infty} D_\tau(P\|Q) = \lim_{\tau \to \infty} \frac{1}{\tau} \sum_{t=1}^{\tau} D(p(y_t|y_{1:t-1})\|q(y_t|y_{1:t-1})) \quad (III.118)
\]

\[
= \lim_{t \to \infty} D(p(y_t|y_{1:t-1})\|q(y_t|y_{1:t-1})) . \quad (III.119)
\]

Running the Kalman filter for \(P\) and \(Q\) to convergence yields filter variables \(\{K_p,F_p\}\) and \(\{K_q,F_q\}\), and conditional covariances \(\{\hat{V}_p,\hat{V}_q\}\). Assuming a stable LDS (i.e. the eigenvalues of \(A\) are within the unit circle), the steady-state covariance \(S(p) = \text{cov}_p(x)\) is the solution to the Lyapunov equation (III.33), which corresponds to the recursive equation (II.8). The estimator variances \(\psi\) and \(\xi\) are the solutions to the Sylvester equations (III.34) and (III.35), which correspond to the recursive equations (III.27) and (III.25). Since \(A_p\) and \(A_q\) are stable, the state means converge to zero, i.e.

\[
\mathbb{E}_p[x] = \mathbb{E}_q[x] = 0 \Rightarrow \mathbb{E}_p[y] = \bar{y}_p, \quad \mathbb{E}_q[y] = \bar{y}_q. \quad (III.120)
\]

The mean of the mismatched state estimator \(\alpha_q\) satisfies

\[
\alpha_q = F_q\alpha_q + A_qK_q(\bar{y}_p - \bar{y}_q), \quad (III.121)
\]

and hence,

\[
\alpha_q = (I - F_q)^{-1}A_qK_q(\bar{y}_p - \bar{y}_q). \quad (III.122)
\]

Finally, for the Mahalanobis distance term

\[
\eta = C_p\mathbb{E}_p(x) - C_q\alpha^{(q)} + \bar{y}_p - \bar{y}_q \quad (III.123)
\]

\[
= (\bar{y}_p - \bar{y}_q) - C_q(I - F_q)^{-1}A_qK_q(\bar{y}_p - \bar{y}_q). \quad (III.124)
\]
III.H.2 Efficiently computing the log determinant

The following theorem illustrates how to efficiently compute the log determinant of a covariance matrix of the form \((CVC^T + R)\) where \(R\) is diagonal.

**Theorem 5.** Suppose \(C \in \mathbb{R}^{m \times n}, V \in \mathbb{S}_+^{n \times n}\), and \(R = \text{diag}(r)\) is a diagonal matrix with \(r \in \mathbb{R}^m\), then

\[
\log |CVC^T + R| = \sum_{i=1}^m \log r_i + \sum_{i=1}^n \log(\lambda_i + 1) \quad \text{(III.125)}
\]

where \(\lambda_i\) are the eigenvalues of \((D_RV D_R^T)\), and \(D = D_QD_R\) is the QR decomposition of \(D = R^{-\frac{1}{2}}C\).

**Proof.** The log-determinant can be computed efficiently via simultaneous diagonalization.

\[
\log |CVC^T + R| = \log \left| R^\frac{1}{2}(R^{-\frac{1}{2}}CVC^T R^{-\frac{1}{2}} + I_m) R^\frac{1}{2} \right| \quad \text{(III.126)}
\]

\[
= \log |R| + \log \left| R^{-\frac{1}{2}}CVC^T R^{-\frac{1}{2}} + I_m \right|. \quad \text{(III.127)}
\]

Let \(D = R^{-\frac{1}{2}}C\) and \(D = D_QD_R\) be the QR decomposition of \(D\), i.e. \(D_Q \in \mathbb{R}^{m \times n}\), \(D_R \in \mathbb{R}^{n \times n}\), and \(D_Q^T D_Q = I_n\). Then the second term in (III.127) becomes

\[
R^{-\frac{1}{2}}CVC^T R^{-\frac{1}{2}} + I_m = DVD^T + I_m \quad \text{(III.128)}
\]

\[
= D_QD_RV D_R^T D_Q^T + I_m. \quad \text{(III.129)}
\]

Defining \(G = D_RV D_R^T\), and its eigen-decomposition \(G = \Phi \Lambda \Phi^T\) where \(\Phi \in \mathbb{R}^{n \times n}\), \(\Phi^T \Phi = I\) and \(\Lambda \in \mathbb{R}^{n \times n}\) is a diagonal matrix of eigenvalues \((\lambda_1, \ldots, \lambda_n)\), then (III.129) becomes

\[
D_QD_RV D_R^T D_Q^T + I_m = D_QGD_Q^T + I_m \quad \text{(III.130)}
\]

\[
= D_Q \Phi \Lambda \Phi^TD_Q^T + I_m \quad \text{(III.131)}
\]

\[
= E \Lambda \hat{E}^T + I_m \quad \text{(III.132)}
\]

\[
= \hat{E} \hat{\Lambda} \hat{E}^T + I_m, \quad \text{(III.133)}
\]
where $E = D_Q \Phi \in \mathbb{R}^{m \times n}$ is orthonormal, i.e. $E^TE = \Phi^T D_Q^TD_Q \Phi = I_n$, and we define the full orthogonal matrix $\hat{E} = \left[ \begin{array}{c} E \\ E' \end{array} \right] \in \mathbb{R}^{m \times m}$ such that $\hat{E}^T \hat{E} = \hat{E} \hat{E}^T = I_m$, and $\hat{\Lambda} = \left[ \begin{array}{cc} \Lambda & 0 \\ 0 & 0 \end{array} \right] \in \mathbb{R}^{m \times m}$. Equations (III.128) and (III.133) are equivalent, hence

$$R^{-\frac{1}{2}} C V C^T R^{-\frac{1}{2}} + I_m = \hat{E} \hat{\Lambda} \hat{E}^T + I_m \quad (III.134)$$

$$\hat{E}^T (R^{-\frac{1}{2}} C V C^T R^{-\frac{1}{2}} + I_m) \hat{E} = \hat{E}^T (\hat{E} \hat{\Lambda} \hat{E}^T + I_m) \hat{E} \quad (III.135)$$

$$\hat{E}^T (R^{-\frac{1}{2}} C V C^T R^{-\frac{1}{2}} + I_m) \hat{E} = \hat{\Lambda} + I_m \quad (III.136)$$

Taking the log-determinant, we have

$$\log \left| \hat{E}^T (R^{-\frac{1}{2}} C V C^T R^{-\frac{1}{2}} + I_m) \hat{E} \right| = \log \left| \hat{\Lambda} + I_m \right| \quad (III.137)$$

$$\log \left| \hat{E}^T \right| \left| R^{-\frac{1}{2}} C V C^T R^{-\frac{1}{2}} + I_m \right| \left| \hat{E} \right| = \log \left| \hat{\Lambda} + I_m \right| \quad (III.138)$$

$$\log \left| R^{-\frac{1}{2}} C V C^T R^{-\frac{1}{2}} + I_m \right| = \log \left| \hat{\Lambda} + I_m \right| \quad (III.139)$$

$$= \sum_{i=1}^{n} \log(\lambda_i + 1) + \sum_{i=n+1}^{m} \log(1) \quad (III.140)$$

$$= \sum_{i=1}^{n} \log(\lambda_i + 1). \quad (III.141)$$

Finally, (III.125) is obtained by substituting (III.141) into (III.127).

**Corollary 1.** Suppose $C \in \mathbb{R}^{m \times n}$ is orthonormal, $V \in S^{n \times n}_+$, and $R = \sigma^2 I$ is an iid covariance matrix, then

$$\log \left| C V C^T + R \right| = m \log \sigma^2 + \sum_{i=1}^{n} \log \left( \frac{\lambda_i}{\sigma^2} + 1 \right), \quad (III.142)$$

where $\lambda_i$ are the eigenvalues of $V$.

**Proof.** Using Theorem 5, we have $D = \frac{1}{\sigma} C$, and its QR decomposition is $D_Q = C$ and $D_R = \frac{1}{\sigma} I_n$. Hence, $(D_R V D_R^T) = \frac{1}{\sigma} V$, which has eigenvalues $(\frac{\lambda_1}{\sigma^2}, \ldots, \frac{\lambda_n}{\sigma^2})$ where $(\lambda_1, \ldots, \lambda_n)$ are the eigenvalues of $V$. \qed
Chapter IV

Mixtures of dynamic textures
IV.A Introduction

One significant limitation of the original dynamic texture model [13] is its inability to provide a perceptual decomposition into multiple regions, each of which belongs to a semantically different visual process: for example, a flock of birds flying in front of a water fountain, highway traffic moving in opposite directions, video containing both smoke and fire, and so forth. One possibility to address this problem is to apply the dynamic texture model locally [15], by splitting the video into a collection of localized spatio-temporal patches, fitting the dynamic texture to each patch, and clustering the resulting models. However, this method, along with other recent proposals [17, 19], lacks some of the attractive properties of the original dynamic texture model: a clear interpretation as a probabilistic generative model for video, and the necessary robustness to operate without manual initialization.

To address these limitations, we note that while the holistic dynamic texture model is not suitable for such scenes, the underlying generative framework is. In fact, co-occurring textures can be accounted for by augmenting the probabilistic generative model with a discrete hidden variable, that has a number of states equal to the number of textures, and encodes which of them is responsible for a given piece of the spatio-temporal video volume. Conditioned on the state of this hidden variable, the video volume is then modeled as a simple dynamic texture. This leads to a natural extension of the dynamic texture model, a mixture of dynamic textures (or mixture of linear dynamical systems), that we study in this chapter. The mixture of dynamic textures is a generative model, where a collection of video sequences (or video patches) are modeled as samples from a set of underlying dynamic textures. It provides a natural probabilistic framework for clustering video, and for video segmentation through the clustering of spatio-temporal patches.

In addition to introducing the dynamic texture mixture as a generative model for video, we report on three main contributions. First, an expectation-maximization (EM) algorithm is derived for maximum-likelihood estimation of the
parameters of a dynamic texture mixture. Second, the relationships between the mixture model and various other models previously proposed, including mixtures of factor analyzers, linear dynamical systems, and switched linear dynamic models, are analyzed. Finally, we demonstrate the applicability of the model to the solution of traditionally difficult vision problems that range from clustering traffic video sequences to segmentation of sequences containing multiple dynamic textures. The remainder of this chapter is organized as follows. In Section IV.B, we formalize the dynamic texture mixture model. In Section IV.C we present the EM algorithm for learning its parameters from training data. In Section IV.D and Section IV.E, we relate it to previous models and discuss its application to video clustering and segmentation. Finally, in Section IV.F we present an experimental evaluation in the context of these applications.

IV.B  Mixtures of dynamic textures

Under the dynamic texture mixture (DTM) model, the observed video sequence $y_{1:\tau} = (y_1, \ldots, y_\tau)$ is sampled from one of $K$ dynamic textures, each having some non-zero probability of occurrence. This is a useful extension for two classes of applications. The first class involves video which is homogeneous at each time instant, but has varying statistics over time. For example, the problem of clustering a set of video sequences taken from a stationary highway traffic camera. While each video will depict traffic moving at homogeneous speed, the exact appearance of each sequence is controlled by the amount of traffic congestion. Different levels of traffic congestion can be represented by $K$ dynamic textures. The second involves inhomogeneous video, i.e. video composed of multiple process that can be individually modeled as dynamic textures of different parameters. For example, in a video scene containing fire and smoke, a random video patch taken from the video will contain either fire or smoke, and a collection of video patches can be represented as a sample from a mixture of two dynamic textures.
Formally, given component priors $\alpha = \{\alpha_1, \ldots, \alpha_K\}$ with $\sum_{j=1}^K \alpha_j = 1$ and dynamic texture components of parameters $\{\Theta_1, \ldots, \Theta_K\}$, a video sequence is drawn by:

1. Sampling a component index $z$ from the multinomial distribution parameterized by $\alpha$.

2. Sampling an observation $y_{1:T}$ from the dynamic texture component of parameters $\Theta_z$.

The probability of a sequence $y_{1:T}$ under this model is

$$p(y_{1:T}) = \sum_{j=1}^K \alpha_j p(y_{1:T} | z = j), \quad \text{(IV.1)}$$

where $p(y_{1:T} | z = j)$ is the class conditional probability of the $j^{th}$ dynamic texture, i.e. the dynamic texture component parameterized by $\Theta_j = \{A_j, Q_j, C_j, R_j, \mu_j, S_j\}$.

The system of equations that define the mixture of dynamic textures is

$$\begin{cases} x_{t+1} = A_z x_t + v_t \\ y_t = C_z x_t + w_t \end{cases} \quad \text{(IV.2)}$$

where the random variable $z \sim \text{multinomial}(\alpha_1, \cdots, \alpha_K)$ signals the mixture component from which the observations are drawn, the initial condition is given by $x_1 \sim \mathcal{N}(\mu_z, S_z)$, and the noise processes by $v_t \sim \mathcal{N}(0, Q_z)$ and $w_t \sim \mathcal{N}(0, R_z)$.

Note that the model adopted in this chapter supports an initial state $x_1$ of arbitrary mean and covariance. This extension produces a richer video model that can capture variability in the initial frame, and is necessary for learning a dynamic texture from multiple video samples with different initial frames (as is the case in clustering and segmentation problems).

The conditional distributions of the states and observations, given the component index $z$, are

$$\begin{align*}
p(x_1 | z) &= G(x_1, \mu_z, S_z), \\
p(x_t | x_{t-1}, z) &= G(x_t, A_z x_{t-1}, Q_z), \\
p(y_t | x_t, z) &= G(y_t, C_z x_t, R_z).
\end{align*} \quad \text{(IV.3-IV.5)}$$
Figure IV.1  Graphical model for a mixture of dynamic textures. The hidden variable $z$ selects the parameters of the remaining nodes.

and the overall joint distribution is

$$p(y_1^T, x_1^T, z) = p(z)p(x_1|z) \prod_{t=2}^{\tau} p(x_t|x_{t-1}, z) \prod_{t=1}^{\tau} p(y_t|x_t, z).$$  \hspace{1cm} (IV.6)

The graphical model for the dynamic texture mixture is presented in Figure IV.1b. Note that, although the addition of the random variable $z$ introduces loops in the graph, exact inference is still tractable because $z$ is connected to all other nodes. Hence, the graph is already moralized and triangulated [55], and the junction tree of Figure IV.1b is equivalent to that of the basic dynamic texture, with the variable $z$ added to each clique. This makes the complexity of exact inference for a mixture of $K$ dynamic textures $K$ times that of the underlying dynamic texture.

IV.C  Parameter estimation using EM

Given a set of i.i.d. video sequences $\{y^{(i)}\}_{i=1}^N$, we would like to learn the parameters $\Theta$ of a mixture of dynamic textures that best fits the data in the maximum-likelihood sense [24], i.e.

$$\Theta^* = \arg \max_{\Theta} \sum_{i=1}^N \log p(y^{(i)}; \Theta).$$  \hspace{1cm} (IV.7)

When the probability distribution depends on hidden variables (i.e. the output of the system is observed, but its state is unknown), the maximum-likelihood solution can be found with the EM algorithm [31]. For the dynamic texture mixture, the
observed information is a set of video sequences \( \{y^{(i)}\}_{i=1}^{N} \), and the missing data consists of: 1) the assignment \( z^{(i)} \) of each sequence to a mixture component, and 2) the hidden state sequence \( x^{(i)} \) that produces \( y^{(i)} \). The EM algorithm is an iterative procedure that alternates between estimating the missing information with the current parameters, and computing new parameters given the estimate of the missing information. In particular, each iteration consists of

\[
\begin{align*}
E - \text{Step} & : Q(\Theta; \hat{\Theta}) = \mathbb{E}_{X,Z|Y; \hat{\Theta}}(\log p(X,Y,Z; \Theta)), \\
M - \text{Step} & : \hat{\Theta}^* = \arg \max_{\Theta} Q(\Theta; \hat{\Theta}),
\end{align*}
\]

where \( p(X,Y,Z; \Theta) \) is the complete-data likelihood of the observations, hidden states, and hidden assignment variables, parameterized by \( \Theta \). To maximize clarity, we only present here the equations of the E and M steps for the estimation of dynamic texture mixture parameters. Their detailed derivation is given in Appendix IV.I. The only assumptions are that the observations are drawn independently and have zero-mean, but the algorithm could be trivially extended to the case of non-zero means. All equations follow the notation of Table IV.1.

### IV.C.1 EM algorithm for mixtures of dynamic textures

Observations are denoted by \( \{y^{(i)}\}_{i=1}^{N} \), the corresponding hidden state variables by \( \{x^{(i)}\}_{i=1}^{N} \), and the hidden assignment variables by \( \{z^{(i)}\}_{i=1}^{N} \). As is usual in the EM literature [31], we introduce a vector \( z_i \in \{0,1\}^K \), such that \( z_{i,j} = 1 \) if and only if \( z^{(i)} = j \). The complete-data log-likelihood is (up to a constant) given
Table IV.1 Notation for EM for mixtures of dynamic textures

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>number of observed sequences.</td>
</tr>
<tr>
<td>$\tau$</td>
<td>length of an observed sequence.</td>
</tr>
<tr>
<td>$K$</td>
<td>number of mixture components.</td>
</tr>
<tr>
<td>$i$</td>
<td>index over the set of observed sequences.</td>
</tr>
<tr>
<td>$j$</td>
<td>index over the components of the mixture.</td>
</tr>
<tr>
<td>$t$</td>
<td>time index of a sequence.</td>
</tr>
<tr>
<td>$y^{(i)}$</td>
<td>the $i^{th}$ observed sequence.</td>
</tr>
<tr>
<td>$y^{(i)}_t$</td>
<td>the observation at time $t$ of $y^{(i)}$.</td>
</tr>
<tr>
<td>$x^{(i)}$</td>
<td>the state sequence corresponding to $y^{(i)}$.</td>
</tr>
<tr>
<td>$x^{(i)}_t$</td>
<td>the state at time $t$ of $x^{(i)}$.</td>
</tr>
<tr>
<td>$z^{(i)}$</td>
<td>the mixture component index for the $i^{th}$ sequence.</td>
</tr>
<tr>
<td>$z_{i,j}$</td>
<td>binary variable which indicates ($z_{i,j}=1$) that $y^{(i)}$ is drawn from component $j$.</td>
</tr>
<tr>
<td>$\alpha_j$</td>
<td>the probability of the $j^{th}$ component, $p(z = j)$.</td>
</tr>
<tr>
<td>$\Theta_j$</td>
<td>the parameters of the $j^{th}$ component.</td>
</tr>
<tr>
<td>$X$</td>
<td>${x^{(i)}}$ for all $i$.</td>
</tr>
<tr>
<td>$Y$</td>
<td>${y^{(i)}}$ for all $i$.</td>
</tr>
<tr>
<td>$Z$</td>
<td>${z^{(i)}}$ for all $i$.</td>
</tr>
</tbody>
</table>

by

$$
\ell(X, Y, Z) = \sum_{i,j} z_{i,j} \log \alpha_j \quad \text{(IV.10)}
$$

$$
- \frac{1}{2} \sum_{i,j} z_{i,j} \text{tr} \left[ S_j^{-1} \left( P^{(i)}_{1,1} - x_1^{(i)} \mu_j^T - \mu_j x_1^{(i)T} + \mu_j^T \mu_j \right) \right]
$$

$$
- \frac{1}{2} \sum_{i,j} \sum_{t=1}^{\tau} \text{tr} \left[ R_j^{-1} \left( y_t^{(i)} y_t^{(i)T} - y_t^{(i)} x_t^{(i)T} C_j^T - C_j x_t^{(i)T} y_t^{(i)} + C_j P^{(i)}_{t,t} C_j^T \right) \right]
$$

$$
- \frac{1}{2} \sum_{i,j} \sum_{t=2}^{\tau} \text{tr} \left[ Q_j^{-1} \left( P^{(i)}_{t,t} - P^{(i)}_{t,t-1} A_j^T - A_j P^{(i)}_{t,t-1}^T + A_j P^{(i)}_{t-1,t-1} A_j^T \right) \right]
$$

$$
- \frac{\tau}{2} \sum_{i,j} z_{i,j} \log |R_j| - \frac{\tau - 1}{2} \sum_{i,j} z_{i,j} \log |Q_j| - \frac{1}{2} \sum_{i,j} z_{i,j} \log |S_j|,
$$
where $P_{t,t}^{(i)} = x_t^{(i)}(x_t^{(i)})^T$ and $P_{t,t-1}^{(i)} = x_t^{(i)}(x_{t-1}^{(i)})^T$. Applying the expectation of (IV.8) to (IV.10) yields the $Q$ function

$$Q(\Theta; \hat{\Theta}) = \sum_j \hat{N}_j \log \alpha_j$$  \hspace{1cm} (IV.11)

\[- \frac{1}{2} \sum_j \text{tr} \left[ R_j^{-1} \left( \Lambda_j - \Gamma_j C_j^T - C_j \Gamma_j^T + C_j \Phi_j C_j^T \right) \right] - \frac{\tau}{2} \sum_j \hat{N}_j \log |R_j| \]

\[- \frac{1}{2} \sum_j \text{tr} \left[ Q_j^{-1} \left( \varphi_j - \Psi_j A_j^T - A_j \Psi_j^T + A_j \phi_j A_j^T \right) \right] - \frac{\tau - 1}{2} \sum_j \hat{N}_j \log |Q_j| \]

\[- \frac{1}{2} \sum_j \text{tr} \left[ S_j^{-1} \left( \eta_j - \xi_j \mu_j^T - \mu_j \xi_j^T + \hat{N}_j \mu_j \mu_j^T \right) \right] - \frac{1}{2} \sum_j \hat{N}_j \log |S_j| , \]

where

\[
\hat{N}_j = \sum_i \hat{z}_{i,j}, \quad \Phi_j = \sum_i \hat{z}_{i,j} \sum_{t=1}^\tau \hat{P}_{t,t|j}^{(i)}, \\
\xi_j = \sum_i \hat{z}_{i,j} \hat{x}_{1|j}^{(i)}, \quad \varphi_j = \sum_i \hat{z}_{i,j} \sum_{t=2}^\tau \hat{P}_{t,t-1|j}^{(i)}, \\
\eta_j = \sum_i \hat{z}_{i,j} \hat{P}_{1,1|j}^{(i)}, \quad \phi_j = \sum_i \hat{z}_{i,j} \sum_{t=2}^\tau \hat{P}_{t-1,t-1|j}^{(i)}, \\
\Psi_j = \sum_i \hat{z}_{i,j} \sum_{t=2}^\tau \hat{P}_{t,t-1|j}, \\
\Lambda_j = \sum_i \hat{z}_{i,j} \sum_{t=1}^\tau y_t^{(i)}(y_t^{(i)})^T, \\
\Gamma_j = \sum_i \hat{z}_{i,j} \sum_{t=1}^\tau y_t^{(i)}(\hat{x}_{1|j}^{(i)})^T
\]

are the aggregates of the expectations

\[
\hat{x}_{1|j}^{(i)} = \mathbb{E}[x_t^{(i)}|y^{(i)}, z^{(i)} = j], \hspace{1cm} (IV.13)
\]

\[
\hat{P}_{t,t|j}^{(i)} = \mathbb{E}[P_{t,t}^{(i)}|y^{(i)}, z^{(i)} = j], \hspace{1cm} (IV.14)
\]

\[
\hat{P}_{t,t-1|j}^{(i)} = \mathbb{E}[P_{t,t-1}^{(i)}|y^{(i)}, z^{(i)} = j], \hspace{1cm} (IV.15)
\]

and the posterior assignment probability is

\[
\hat{z}_{i,j} = p(z^{(i)} = j|y^{(i)}) = \frac{\alpha_j p(y^{(i)}|z^{(i)} = j)}{\sum_{k=1}^K \alpha_k p(y^{(i)}|z^{(i)} = k)}.
\]

(IV.16)

Hence, the E-step consists of computing the conditional expectations (IV.13)-(IV.16), and can be implemented efficiently with the Kalman smoothing filter (see Section II.C), which estimates the mean and covariance of the state $x^{(i)}$ conditioned
on the observation $y^{(i)}$ and $z^{(i)} = j$,

$$\hat{x}_{t_i}^{(i)} = \mathbb{E}[x_t | y^{(i)}, z^{(i)} = j], \quad \text{(IV.17)}$$

$$\hat{V}_{t,t}^{(i)} = \text{cov}(x_t | y^{(i)}, z^{(i)} = j), \quad \text{(IV.18)}$$

$$\hat{V}_{t,t-1}^{(i)} = \text{cov}(x_t, x_{t-1} | y^{(i)}, z^{(i)} = j). \quad \text{(IV.19)}$$

The second-order moments of (IV.14) and (IV.15) are then calculated as $\hat{P}_{t,t}^{(i)} = \hat{V}_{t,t}^{(i)} + \hat{x}_{t_i}^{(i)}(\hat{x}_{t_i}^{(i)})^T$ and $\hat{P}_{t,t-1}^{(i)} = \hat{V}_{t,t-1}^{(i)} + \hat{x}_{t_i}^{(i)}(\hat{x}_{t_i}^{(i-1)})^T$. Finally, the data likelihood $p(y^{(i)} | z^{(i)} = j)$ is computed using the “innovations” form of the log-likelihood (again, see Section II.C).

In the M-step, the dynamic texture parameters are updated according to (IV.9), resulting in the following update step for each mixture component $j$,

$$C_j^* = \Gamma_j(\Phi_j)^{-1}, \quad R_j^* = \frac{1}{\tau N_j} \left( A_j - C_j^* \Gamma_j \right),$$

$$A_j^* = \Psi_j(\phi_j)^{-1}, \quad Q_j^* = \frac{1}{(r-1)N_j} \left( \varphi_j - A_j^* \Psi_j^T \right),$$

$$\mu_j^* = \frac{1}{N_j} \xi_j, \quad S_j^* = \frac{1}{N_j} \eta_j - \mu_j^*(\mu_j^*)^T,$$

$$\alpha_j^* = \frac{N_j}{N}.$$

A summary of EM for the mixture of dynamic textures is presented in Algorithm 4. The E-step relies on the Kalman smoothing filter to compute: 1) the expectations of the hidden state variables $x_t$, given the observed sequence $y^{(i)}$ and the component assignment $z^{(i)}$; and 2) the likelihood of observation $y^{(i)}$ given the assignment $z^{(i)}$. The M-step then computes the maximum-likelihood parameter values for each dynamic texture component $j$, by averaging over all sequences $\{y^{(i)}\}_{i=1}^{N}$, weighted by the posterior probability of assigning $z^{(i)} = j$.

### IV.C.2 Initialization strategies

It is known that the accuracy of parameter estimates produced by EM is dependent on how the algorithm is initialized. In the remainder of this section, we present three initialization strategies that we have empirically found to be effective for learning mixtures of dynamic textures.
Algorithm 4 EM algorithm for mixtures of dynamic textures

**Input:** \( N \) sequences \( \{y^{(i)}\}_{i=1}^{N} \), number of components \( K \).

1. Initialize \( \{\Theta_j, \alpha_j\} \) for \( j = 1 \) to \( K \).

2. **repeat**
   - **Expectation Step**
     - **for** \( i = \{1, \ldots, N\} \) and \( j = \{1, \ldots, K\} \) **do**
       - Compute the expectations (IV.13-IV.16) with the Kalman smoothing filter (Section II.C) on \( y^{(i)} \) and \( \Theta_j \).
     - **end for**
   - **Maximization Step**
     - **for** \( j = 1 \) to \( K \) **do**
       - Compute aggregate expectations (IV.12).
       - Compute new parameters \( \{\Theta_j, \alpha_j\} \) with (IV.20).
     - **end for**
   - **until** convergence

**Output:** \( \{\Theta_j, \alpha_j\}_{j=1}^{K} \).

**Initial seeding**

If an initial clustering of the data is available (e.g. by specification of initial contours for segmented regions), then each mixture component is learned by application of the least-squares method to each of the initial clusters.

**Random trials**

Several trials of EM are run with different random initializations, and the parameters which best fit the data, in the maximum-likelihood sense, are selected. For each EM trial, each mixture component is initialized by application of the least-squares method to a randomly selected example from the dataset.
Component splitting

The EM algorithm is run with an increasing number of mixture components, a common strategy in the speech-recognition community [56]:

1. Run the EM algorithm with $K = 1$ mixture components.

2. Duplicate a mixture component, and perturb the new component’s parameters.

3. Run the EM algorithm, using the new mixture model as initialization.

4. Repeat Steps 2 and 3 until the desired number of components is reached.

For the mixture of dynamic textures, we use the following selection and perturbation rules: 1) the mixture component with the largest misfit in the state-space (i.e. the mixture component with the largest eigenvalue of $Q$) is selected for duplication; and 2) the principal component whose coefficients have the largest initial variance (i.e. the column of $C$ associated with the largest variance in $S$) is scaled by 1.01.

IV.D Connections to the literature

Although novel as a tool for modeling video, with application to problems such as clustering and segmentation, the mixture of dynamic textures and the proposed EM algorithm are related to various previous works in adaptive filtering, statistics and machine learning, time-series clustering, and video segmentation.

IV.D.1 Adaptive filtering and control

In the control-theory literature, [57] proposes an adaptive filter based on banks of Kalman filters running in parallel, where each Kalman filter models a mode of a physical process. The output of the adaptive filter is the average of the outputs of the individual Kalman filters, weighted by the posterior probability that the observation was drawn from the filter. This is an inference procedure for
the hidden state of a mixture of dynamic textures conditioned on the observation. The key difference with respect to this work is that, while [57] focuses on inference on the mixture model with known parameters, this work focuses on learning the parameters of the mixture model. The work of [57] is the forefather of other multiple-model based methods in adaptive estimation and control [58, 59, 60], but none address the learning problem.

IV.D.2 Models proposed in statistics and machine learning

For a single component \((K = 1)\) and a single observation \((N = 1)\), the EM algorithm for the mixture of dynamic textures reduces to the classical EM algorithm for learning an LDS (Section II.D.1). The LDS is a generalization of the factor analysis model [25], a statistical model which explains an observed vector as a combination of measurements which are driven by independent factors. Similarly the mixture of dynamic textures is a generalization of the mixture of factor analyzers\(^1\), and the EM algorithm for a dynamic texture mixture reduces to the EM algorithm for learning a mixture of factor analyzers [61].

The dynamic texture mixture is also related to “switching” linear dynamical models, where the parameters of a LDS are selected via a separate Markovian switching variable as time progresses. Variations of these models include [62, 63] where only the observation matrix \(C\) switches, [64, 65, 66] where the state parameters switch \((A\) and \(Q\)), and [67, 68] where the observation and state parameters switch \((C, R, A,\) and \(Q)\). These models have one state variable that evolves according to the active system parameters at each time step. This makes the switching model a mixture of an exponentially increasing number of LDSs with time-varying parameters.

In contrast to switching models with a single state variable, the switching state-space model proposed in [69] switches the observed variable between the

---

\(^1\)Restricting the LDS parameters \(S = Q = I_n, \mu = 0, A = 0,\) and \(R\) as a diagonal matrix yields the factor analysis model. Similarly for the mixture of dynamic textures, setting \(S_j = Q_j = I\) and \(A_j = 0\) for each factor analysis component, and \(\tau = 1\) (since there are no temporal dynamics) yields the mixture of factor analyzers.
output of different LDSs at each time step. Each LDS has its own observation matrix and state variable, which evolves according to its own system parameters. The difference between the switching state-space model and the mixture of dynamic textures is that the switching state-space model can switch between LDS outputs at each time step, whereas the mixture of dynamic textures selects an LDS only once at time \( t = 1 \), and never switches from it. Hence, the mixture of dynamic textures is similar to a special case of the switching state-space model, where the initial probabilities of the switching variable are the mixture component probabilities \( \alpha_j \), and the Markovian transition matrix of the switching variable is equal to the identity matrix. The ability of the switching state-space model to switch at each time step results in a posterior distribution that is a Gaussian mixture with a number of terms that increases exponentially with time [69].

While the mixture of dynamic textures is closely related to both switching LDS models and the model of [69], the fact that it selects only one LDS per observed sequence makes the posterior distribution a mixture of a constant number of Gaussians. This key difference has consequences of significant practical importance. First, when the number of components increases exponentially (as is the case for models that involve switching), exact inference becomes intractable, and the EM-style of learning requires approximate methods (e.g. variational approximations) which are, by definition, sub-optimal. Second, because the exponential increase in the number of degrees of freedom is not accompanied by an exponential increase in the amount of available data (which only grows linearly with time), the difficulty of the learning problem increases with sequence length. None of these problems affect the dynamic texture mixture, for which exact inference is tractable, allowing the derivation of the exact EM algorithm presented above.

**IV.D.3 Time-series clustering**

Although we apply the mixture of dynamic textures to video, the model is general and can be used to cluster any type of time-series. When compared to
the literature in this field [70], the mixture of dynamic textures can be categorized as a model-based method for clustering multivariate continuous-valued time-series data. Two alternative methods are available for clustering this type of data, both based on the K-means algorithm with distance (or similarity) measures suitable for time-series: 1) [71] measures the distance between two time-series with the KL divergence or the Chernoff measure, which are estimated non-parametrically in the spectral domain; and 2) [72] measures similarity by comparing the PCA subspaces and the means of the time-series, but disregards the dynamics of the time-series. The well known connection between EM and K-means makes these algorithms somewhat related to the dynamic texture mixture, but they lack a precise probabilistic interpretation. Finally, the dynamic texture mixture is related to the ARMA mixture proposed in [73]. The main differences are that the ARMA mixture 1) only models univariate data, and 2) does not utilize a hidden state model. On the other hand, the ARMA mixture supports high-order Markov models, while the dynamic texture mixture is based on a first-order Markovian assumption.

IV.D.4 Video segmentation

The idea of applying dynamic texture representations to the segmentation of video has previously appeared in the video literature. In fact, some of the inspiration for our work was the promise shown for temporal texture segmentation (e.g. smoke and fire) by the dynamic texture model. For example, [15] segments video by clustering patches of dynamic texture using the level-sets framework and the Martin distance. More recently, [19] clusters pixel-intensities (or local texture features) using auto-regressive processes (AR) and level-sets, and [17] segments video by clustering pixels with similar trajectories in time, using generalized PCA (GPCA). While these methods have shown promise, they do not exploit the probabilistic nature of the dynamic texture representation for the segmentation itself. On the other hand, the segmentation algorithms proposed in the following section are statistical procedures that leverage on the mixture of dynamic texture
to perform optimal inference. This results in greater robustness to variations due to the stochasticity of the video appearance and dynamics, leading to superior segmentation results, as will be demonstrated in Sections IV.E and IV.F.

IV.E Applications to video clustering and motion segmentation

Like any other probabilistic model, the dynamic texture has a large number of potential application domains, many of which extend well beyond the field of computer vision (e.g. modeling of high-dimensional time series for financial applications, weather forecasting, etc.). In this chapter, we concentrate on vision applications, where mixture models are frequently used to solve problems such as clustering [74, 37], background modeling [75], image segmentation and layering [76, 6, 77, 78, 79, 80], or retrieval [81, 80]. The dynamic texture mixture extends this class of methods to problems involving video of particle ensembles subject to stochastic motion. We consider, in particular, the problems of clustering and segmentation.

IV.E.1 Video clustering

Video clustering can be a useful tool to uncover high-level patterns of structure in a video stream, e.g. recurring events, events of high and low probability, outlying events, etc. These operations are of great practical interest for some classes of particle-ensemble video, such as those which involve understanding video acquired in crowded environments. In this context, video clustering has application to problems such as surveillance, novelty detection, event summarization, or remote monitoring of various types of environments. It can also be applied to the entries of a video database in order to automatically create a taxonomy of video classes that can then be used for database organization or video retrieval. Under the mixture of dynamic textures representation, a set of video sequences is clus-
tered by first learning the mixture that best fits the entire collection of sequences, and then assigning each sequence to the mixture component with largest posterior probability of having generated it, i.e. by labeling sequence \( y^{(i)} \) with

\[
\ell_i = \arg \max_j \log p(y^{(i)}|z^{(i)} = j) + \log \alpha_j. \tag{IV.21}
\]

### IV.E.2 Motion segmentation

Video segmentation addresses the problem of decomposing a video sequence into a collection of homogeneous regions. While this has long been known to be solvable with mixture models and the EM algorithm [37, 6, 7, 76, 77], the success of the segmentation operation depends on the ability of the mixture model to capture the dimensions along which the video is statistically homogeneous. For spatio-temporal textures (e.g. video of smoke and fire), traditional mixture-based motion models are not capable of capturing these dimensions, due to their inability to account for the stochastic nature of the underlying motion. The mixture of dynamic textures extends the application of mixture-based segmentation algorithms to video composed of spatio-temporal textures. As in most previous mixture-based approaches to video segmentation, the process consists of two steps. The mixture model is first learned, and the video is then segmented by assigning video locations to mixture components.

In the learning stage, the video is first represented as a bag of patches. For spatio-temporal texture segmentation, a patch of dimensions \( p \times p \times q \) is extracted from each location in the video sequence (or along a regularly spaced grid)\(^2\) where \( p \) and \( q \) should be large enough to capture the distinguishing characteristics of the various components of the local motion field. Note that this is unlike methods that model the changes of appearance of single pixels (e.g. [17] and the AR model of [19]) and, therefore, have no ability to capture the spatial coherence of the local motion field. If the segmentation boundaries are not expected to change over

\(^2\)Although overlapping patches violate the independence assumption, they work well in practice and are commonly adopted in computer vision.
time, \(q\) can be set to the length of the video sequence. The set of spatio-temporal patches is then clustered with recourse to the EM algorithm of Section IV.C. The second step, segmentation, scans the video locations sequentially. At each location, a patch is extracted, and assigned to one of the mixture components, according to (IV.21). The location is then declared to belong to the segmentation region associated with that cluster.

It is interesting to note that the mixture of dynamic textures can be used to efficiently segment very long video sequences, by first learning the mixture model on a short training sequence (e.g. a clip from the long sequence), and then segmenting the long sequence with the learned mixture. The segmentation step only requires computing the patch log-likelihoods under each mixture component, i.e. (II.23). Since the conditional covariances \(\hat{V}_{t|t-1}\) and \(\hat{V}_{t|t}\), and the Kalman filter gains \(K_t\) do not depend on the observation \(y_t\), they can be pre-computed. The computational steps required for the data-likelihood of a single patch therefore reduce to computing, \(\forall t = \{1, \ldots, \tau\}\),

\[
\hat{x}_{t|t-1} = A\hat{x}_{t-1|t-1}, \tag{IV.22}
\]

\[
\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(y_t - C\hat{x}_{t|t-1}), \tag{IV.23}
\]

\[
p(y_t|y_{1:t-1}) = -\frac{1}{2} \log |M| - \frac{m}{2} \log(2\pi) \tag{IV.24}
\]

\[-\frac{1}{2}(y_t - C\hat{x}_{t|t-1})^T M^{-1}(y_t - C\hat{x}_{t|t-1}).\]

where \(M = C\hat{V}_{t|t-1}C^T + R\). Hence, computing the data-likelihood under one mixture component of a single patch requires \(O(5\tau)\) matrix-vector multiplications. For a mixture of \(K\) dynamic textures and \(N\) patches, the computation is \(O(5\tau KN)\) matrix-vector multiplications.

**IV.F Experimental evaluation**

The performance of the mixture of dynamic textures was evaluated with respect to various applications: 1) clustering of time-series data, 2) clustering
of highway traffic video, and 3) motion segmentation of both synthetic and real
video sequences. In all cases, performance was compared to a representative of the
state-of-the-art for these application domains. The initialization strategies from
Section IV.C.2 were used. The observation noise was assumed to be independent
and identically distributed (i.e. $R = \sigma^2 I_m$), the initial state covariance $S$ was
assumed to be diagonal, and the covariance matrices $Q$, $S$, and $R$ were regularized
by enforcing a lower bound on their eigenvalues. Videos of the results from all
experiments are available in the supplemental material [52].

IV.F.1 Time-series clustering

We start by presenting results of a comparison of the dynamic texture
mixture with several multivariate time-series clustering algorithms. To enable an
evaluation based on clustering ground truth, this comparison was performed on
a synthetic time-series dataset, generated as follows. First, the parameters of $K$
LDSs, with state-space dimension $n = 2$ and observation-space dimension $m = 10$,
were randomly generated according to

\[
\begin{align*}
\mu &\sim \mathcal{U}_n(-5, 5), & S &\sim \mathcal{W}(I_n, n), & C &\sim \mathcal{N}_{m,n}(0, 1), \\
Q &\sim \mathcal{W}(I_n, n), & \lambda_0 &\sim \mathcal{U}_1(0.1, 1), & A_0 &\sim \mathcal{N}_{n,n}(0, 1), \\
\sigma^2 &\sim \mathcal{W}(1, 2), & R &= \sigma^2 I_m, & A &= \lambda_0 A_0/\lambda_{\text{max}}(A_0),
\end{align*}
\]

where $\mathcal{N}_{m,n}(\mu, \sigma^2)$ is a distribution on $\mathbb{R}^{m \times n}$ matrices with each entry distributed
as $\mathcal{N}(\mu, \sigma^2)$, $\mathcal{W}(\Sigma, d)$ is a Wishart distribution with covariance $\Sigma$ and $d$ degrees of
freedom, $\mathcal{U}_d(a, b)$ is a distribution on $\mathbb{R}^d$ vectors with each coordinate distributed
uniformly between $a$ and $b$, and $\lambda_{\text{max}}(A_0)$ is the magnitude of the largest eigenvalue
of $A_0$. Note that $A$ is a random scaling of $A_0$ such that the system is stable (i.e.
the poles of $A$ are within the unit circle). A time-series dataset was generated
by sampling 20 time-series of length 50 from each of the $K$ LDSs. Finally, each
time-series sample was normalized to have zero temporal mean.

The data was clustered using the mixture of dynamic textures (Dy-
texMix), and three multivariate-series clustering algorithms from the time-series
literature. The latter are based on variations of K-means for various similarity measures: PCA subspace similarity (Singhal) [72]; the KL divergence (KakKL) [71]; and the Chernoff measure (KakCh) [71]. As a baseline, the data was also clustered with K-means [37] using the Euclidean distance (K-means) and the cosine similarity (K-means-c) on feature vectors formed by concatenating each time-series. The correctness of a clustering is measured quantitatively using the Rand index [82] between the clustering and the ground-truth. Intuitively, the Rand index corresponds to the probability of pair-wise agreement between the clustering and the ground-truth, i.e. the probability that the assignment of any two items will be correct with respect to each other (in the same cluster, or in different clusters). For each algorithm, the average Rand index was computed for each value of $K = \{2, 3, \ldots, 8\}$, by averaging over 100 random trials of the clustering experiment. We refer to this synthetic experiment setup as “SyntheticA”. The clustering algorithms were also tested on two variations of the experiment based on time-series that were more difficult to cluster. In the first (SyntheticB), the $K$ random LDSs were forced to share the same observation matrix ($C$), therefore forcing all time-series to be defined in similar subspaces. In the second (SyntheticC), the LDSs had large observation noise, i.e. $\sigma^2 \sim \mathcal{W}(16, 2)$. Note that these variations are typically expected in video clustering problems. For example, in applications where the appearance component does not change significantly between clusters (e.g. highway video with varying levels of traffic), all the video will span similar image subspaces.

Figure IV.2 presents the results obtained with the six clustering algorithms on the three experiments, and Table IV.2 shows the overall Rand index, computed by averaging over $K$. In SyntheticA (Figure IV.2a), Singhal, KakCh, and DytexMix achieved comparable performance (overall Rand of 0.995, 0.991, and 0.991) with Singhal performing slightly better. On the other hand, it is clear that the two baseline algorithms are not suitable for clustering time-series data, albeit there is significant improvement when using K-means-c (0.831) rather than
Figure IV.2 Time-series clustering results on three synthetic problems: (a) SyntheticA; (b) SyntheticB; and (c) SyntheticC. Plots show the Rand index versus the number of clusters ($K$) for six time-series clustering algorithms.
Table IV.2 Overall Rand index on the three synthetic experiments for six clustering algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>SyntheticA</th>
<th>SyntheticB</th>
<th>SyntheticC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DytexMix</td>
<td>0.991</td>
<td><strong>0.995</strong></td>
<td><strong>0.993</strong></td>
</tr>
<tr>
<td>Singhal [72]</td>
<td><strong>0.995</strong></td>
<td>0.865</td>
<td>0.858</td>
</tr>
<tr>
<td>KakKL [71]</td>
<td>0.946</td>
<td>0.977</td>
<td>0.960</td>
</tr>
<tr>
<td>KakCh [71]</td>
<td>0.991</td>
<td>0.985</td>
<td>0.958</td>
</tr>
<tr>
<td>K-means</td>
<td>0.585</td>
<td>0.624</td>
<td>0.494</td>
</tr>
<tr>
<td>K-means-c</td>
<td>0.831</td>
<td>0.781</td>
<td>0.746</td>
</tr>
</tbody>
</table>

K-means (0.585). In SyntheticB (Figure IV.2b), the results were different: the DytexMix performed best (0.995), closely followed by KakCh and KakKL (0.985 and 0.977). On the other hand, Singhal did not perform well (0.865) because all the time-series have similar PCA subspaces. Finally, in SyntheticC (Figure IV.2c), the DytexMix repeated the best performance (0.993), followed by KakKL and KakCh (0.960 and 0.958), with Singhal performing the worst again (0.858). In this case, the difference between the performance of DytexMix and those of KakKL and KakCh was significant. This can be explained by the robustness of the former to observation noise, a property not shared by the latter due to the fragility of non-parametric estimation of spectral matrices.

In summary, these results show that the mixture of dynamic textures performs similarly to state-of-the-art methods, such as [72] and [71], when clusters are well separated. It is, however, more robust against occurrences that increase the amount of cluster overlap, which proved difficult for the other methods. Such occurrences include 1) time-series defined in similar subspaces, and 2) time-series with significant observation noise, and are common in video clustering applications.

IV.F.2 Video clustering

To evaluate its performance in problems of practical significance, the dynamic texture mixture was used to cluster video of vehicle highway traffic. Clustering was performed on 253 video sequences collected by the Washington Department
Figure IV.3 Clustering of traffic video: (a) six typical sequences from the five clusters; (b) speed measurements from the highway loop sensor over time; and the clustering index over time for (c) 2 clusters, and (d) 5 clusters.
of Transportation (WSDOT) on interstate I-5 in Seattle, Washington [53]. Each video clip is 5 seconds long and the collection spanned about 20 hours over two days. The video sequences were converted to grayscale, and normalized to have size $48 \times 48$ pixels, zero mean, and unit variance.

The mixture of dynamic textures was used to organize this dataset into 5 clusters. Figure IV.3a shows six typical sequences for each of the five clusters. These examples, and further analysis of the sequences in each cluster, reveal that the clusters are in agreement with classes frequently used in the perceptual categorization of traffic: light traffic (spanning 2 clusters), medium traffic, slow traffic, and stopped traffic (“traffic jam”). Figure IV.3b, Figure IV.3c, and Figure IV.3d show a comparison between the temporal evolution of the cluster index and the average traffic speed. The latter was measured by the WSDOT with an electromagnetic sensor (commonly known as a loop sensor) embedded in the highway asphalt, near the camera. The speed measurements are shown in Figure IV.3b and the temporal evolution of the cluster index is shown for $K = 2$ (Figure IV.3c) and $K = 5$ (Figure IV.3d). Unfortunately, a precise comparison between the speed measurements and the video is not possible because the data originate from two separate systems, and the video data is not time-stamped with fine enough precision. Nonetheless, it is still possible to examine the correspondence between the speed data and the video clustering. For $K = 2$, the algorithm forms two clusters that correspond to fast-moving and slow-moving traffic. Similarly, for $K = 5$, the algorithm creates two clusters for fast-moving traffic, and three clusters for slow-moving traffic (which correspond to medium, slow, and stopped traffic).

**IV.F.3 Motion segmentation**

Several experiments were conducted to evaluate the usefulness of dynamic texture mixtures for video segmentation. In all cases, two initialization methods were considered: the manual specification of a rough initial segmentation contour (referred to as DytexMixIC), and the component splitting strategy of
Table IV.3  Videos in motion segmentation experiment. $K$ is the number of clusters.

<table>
<thead>
<tr>
<th>video</th>
<th>size</th>
<th>patch size</th>
<th>$K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocean-steam</td>
<td>220×220×140</td>
<td>7×7×140</td>
<td>2</td>
</tr>
<tr>
<td>ocean-appearance</td>
<td>160×110×120</td>
<td>5×5×120</td>
<td>2</td>
</tr>
<tr>
<td>ocean-dynamics</td>
<td>160×110×120</td>
<td>5×5×120</td>
<td>2</td>
</tr>
<tr>
<td>ocean-fire</td>
<td>160×110×120</td>
<td>5×5×5</td>
<td>2</td>
</tr>
<tr>
<td>synthdb</td>
<td>160×110×60</td>
<td>5×5×60</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>highway traffic</td>
<td>160×112×51</td>
<td>5×5×51</td>
<td>4</td>
</tr>
<tr>
<td>bridge traffic</td>
<td>160×113×51</td>
<td>5×5×51</td>
<td>4</td>
</tr>
<tr>
<td>fountains</td>
<td>160×110×75</td>
<td>5×5×75</td>
<td>3</td>
</tr>
<tr>
<td>pedestrian 1</td>
<td>238×158×200</td>
<td>7×7×20</td>
<td>2</td>
</tr>
<tr>
<td>pedestrian 2</td>
<td>238×158×200</td>
<td>7×7×20</td>
<td>2</td>
</tr>
</tbody>
</table>

Section IV.C.2 (DytexMixCS). The segmentation results are compared with those produced by several segmentation procedures previously proposed in the literature: the level-sets method of [19] using Ising models (Ising); generalized PCA (GPCA) [17], and two algorithms representative of the state-of-the-art for traditional optical-flow based motion segmentation. The first method (NormCuts) is based on normalized cuts [83] and the “motion profile” representation proposed in [83, 84]\(^3\). The second (OpFlow) represents each pixel as a feature-vector containing the average optical-flow over a 5×5 window, and clusters the feature-vectors using the mean-shift algorithm [85].

Preliminary evaluation revealed various limitations of the different techniques. For example, the optical flow methods cannot deal with the stochasticity of microscopic textures (e.g. water and smoke), performing much better for textures composed of non-microscopic primitives (e.g. video of crowded scenes composed of objects, such as cars or pedestrians, moving at a distance). On the other hand, level-sets and GPCA perform best for microscopic textures. Due to these limitation, and for the sake of brevity, we limit the comparisons that follow to the methods that performed best for each type of video under consideration. In some cases, the experiments were also restricted by implementation constraints. For ex-

\(^3\)For this representation, we used a patch of size 15×15 and a motion profile neighborhood of 5×5.
Figure IV.4 Segmentation of synthetic video from [15]: (a) ocean-steam; (b) ocean-appearance; and (c) ocean-dynamics. The first column shows a video frame and the initial contour, and the remaining columns show the segmentations from DytexMixIC, Ising [19], GPCA [17], and Martin distance (from [15]).

ample, the available implementations of the level-sets method can only be applied to video composed of two textures.

In all experiments the video are grayscale, and the video patches are either $5 \times 5$ or $7 \times 7$ pixels, depending on the image size (see Table IV.3 for more details). The patches are normalized to have zero temporal mean, and unit variance. Unless otherwise specified, the dimension of the state-space is $n = 10$. Finally, all segmentations are post-processed with a $5 \times 5$ “majority” smoothing filter.

**Segmentation of synthetic video**

We start with the four synthetic sequences studied in [15]: 1) steam over an ocean background (ocean-steam); 2) ocean with two regions rotated by ninety degrees, i.e. regions of identical dynamics but different appearance (ocean-appearance); 3) ocean with two regions of identical appearance but different dynamics (ocean-dynamics); and 4) fire superimposed on an ocean background (ocean-fire). The segmentations of the first three videos are shown in Figure IV.4. Quali-
Figure IV.5 Segmentation of ocean-fire [15]: (a) video frames; and the segmentation using (b) DytexMixIC; and (c) Ising [19].

Qualitatively, the segmentations produced by DytexMixIC and Ising ($n = 2$) are similar, with Ising performing slightly better with respect to the localization of the segmentation borders. While both these methods improve on the segmentations of [15], GPCA can only segment ocean-steam, failing on the other two sequences. We next consider the segmentation of a sequence containing ocean and fire (Figure IV.5a). This sequence is more challenging because the boundary of the fire region is not stationary (i.e. the region changes over time as the flame evolves). Once again, DytexMixIC ($n = 2$) and Ising ($n = 2$) produce comparable segmentations (Figure IV.5b and Figure IV.5c), and are capable of tracking the flame over time. We were not able to produce any meaningful segmentation with GPCA on this sequence.

In summary, both DytexMixIC and Ising perform qualitatively well on the sequences from [15], but GPCA does not. The next section will compare these algorithms quantitatively, using a much larger database of synthetic video.

**Segmentation of synthetic video database**

The segmentation algorithms were evaluated quantitatively on a database of 300 synthetic sequences. These consist of three groups of one hundred videos, with each group generated from a common ground-truth template, for $K = \{2, 3, 4\}$. 
Table IV.4: The best average Rand index for each segmentation algorithm on the synthetic texture database. The order of the model \((n)\) is shown in parenthesis.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(K = 2)</th>
<th>(K = 3)</th>
<th>(K = 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DytexMixIC</td>
<td>0.915 (17)</td>
<td>0.853 (15)</td>
<td>0.868 (15)</td>
</tr>
<tr>
<td>DytexMixCS</td>
<td>0.915 (20)</td>
<td>0.825 (10)</td>
<td>0.835 (15)</td>
</tr>
<tr>
<td>Ising [19]</td>
<td>0.879 (05)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>AR [19]</td>
<td>0.690 (02)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>AR0 [19]</td>
<td>0.662 (05)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>GPCA [17]</td>
<td>0.548 (02)</td>
<td>0.554 (17)</td>
<td>0.549 (10)</td>
</tr>
<tr>
<td>Baseline Rand.</td>
<td>0.607</td>
<td>0.523</td>
<td>0.501</td>
</tr>
<tr>
<td>Baseline Init</td>
<td>0.600</td>
<td>0.684</td>
<td>0.704</td>
</tr>
</tbody>
</table>

The segments were randomly selected from a set of 12 textures, which included grass, sea, boiling water, moving escalator, fire, steam, water, and plants. Examples from the database can be seen in Figure IV.7a and Figure IV.8a, along with the initial contours provided to the segmentation algorithms.

All sequences were segmented with DytexMixIC, DytexMixCS, and GPCA [17]. Due to implementation limitations, the algorithms of [19], Ising, AR, and AR0 (AR without mean) were applied only for \(K = 2\). All methods were run with different orders of the motion models \((n)\), while the remaining parameters for each algorithm were fixed throughout the experiment. Performance was evaluated with the Rand index [82] between segmentation and ground-truth. In addition, two baseline segmentations were included: 1) “Baseline Random”, which randomly assigns pixels; and 2) “Baseline Init”, which is the initial segmentation (i.e. initial contour) provided to the algorithms.

Figure IV.6a shows the average Rand index resulting from the segmentation algorithms for different orders \(n\), and Table IV.4 presents the best results for each algorithm. For all \(K\), DytexMixIC achieved the best overall performance, i.e. largest average Rand index. For \(K = 2\), Ising also performs well (0.879), but is inferior to DytexMixIC (0.916). The remaining algorithms performed poorly, with GPCA performing close to random pixel assignment.

While the average Rand index quantifies the overall performance on the
Figure IV.6 Results on the synthetic database: a) average Rand index versus the order of the motion model ($n$); and b) segmentation precision for the best $n$ for each algorithm. Each row presents the results for 2, 3, or 4 segments in the database.
database, it does not provide insight on the characteristics of the segmentation failures, i.e. whether there are many small errors, or a few gross ones. To overcome this limitation, we have also examined the segmentation precision of all algorithms. Given a threshold $\theta$, segmentation precision is defined as the percentage of segmentations deemed to be correct with respect to the threshold, i.e. the percentage with Rand index larger than $\theta$. Figure IV.6b plots the precision of the segmentation algorithms for different threshold levels. One interesting observation can be made when $K = 2$. In this case, for a Rand threshold of 0.95 (corresponding to tolerance of about 1 pixel error around the border), the precisions of DytexMixIC and Ising are, respectively, 73% and 51%. On the other hand, for very high thresholds (e.g. 0.98), Ising has a larger precision (31% versus 14% of DytexMixIC). This suggests that Ising is very good at finding the exact boundary when nearly perfect segmentations are possible, but is more prone to dramatic segmentation failures. On the other hand, DytexMixIC is more robust, but not as precise near borders.

An example of these properties is shown in Figure IV.7. The first column presents a sequence for which both methods work well, while the second column shows an example where Ising is more accurate near the border, but where DytexMixIC still finds a good segmentation. The third and fourth columns present examples where DytexMixIC succeeds and Ising fails (e.g. in the fourth column, Ising confuses the bright escalator edges with the wave ripples). Finally, both methods fail in the fifth column, due to the similarity of the background and foreground water.

With respect to initialization, it is clear from Figure IV.6 and Table IV.4 that, even in the absence of an initial contour, the mixture of dynamic textures (DytexMixCS) outperforms all other methods considered, including those that require an initial contour (e.g. Ising) and those that do not (e.g. GPCA). Again, this indicates that video segmentation with the dynamic texture mixture is quite robust. Comparing DytexMixIC and DytexMixCS, the two methods achieve equivalent performance for $K = 2$, with DytexMixIC performing slightly better.
Figure IV.7 Segmentation examples from the synthetic texture database ($K = 2$): a) a video frame and the initial contour; segmentation with: b) DytexMixIC; and c) Ising [19]. The Rand index ($r$) of each segmentation is shown above the image.

for $K = 3$ and $K = 4$. With multiple textures, manual specification of the initial contour reduces possible ambiguities due to similarities between pairs of textures. An example is given in the third column of Figure IV.8, where the two foreground textures are similar, while the background texture could perceptually be split into two regions (particles moving upward and to the right, and those moving upward and to the left). In the absence of initial contour, DytexMixCS prefers this alternative interpretation. Finally, the first two columns of Figure IV.8 show examples where both methods perform well, while in the fourth and fifth columns both fail (e.g. when two water textures are almost identical).

In summary, the quantitative results on a large database of synthetic video textures indicate that the mixture of dynamic textures (both with or without initial contour specification) is superior to all other video texture segmentation algorithms considered. While Ising also achieves acceptable performance, it has two limitations: 1) it requires an initial contour, and 2) it can only segment video composed of two regions. Finally, AR and GPCA do not perform well on this database.
Figure IV.8  Segmentation examples from the synthetic texture database ($K = 3$ and $K = 4$): a) a video frame and the initial contour; b) segmentation using DytxMixIC; and c) DytxMixCS. The Rand index ($r$) of each segmentation is shown above the image.

Segmentation of real video

We finish with segmentation results on five real video sequences, depicting a water fountain, highway traffic, and pedestrian crowds. While a precise evaluation is not possible, because there is no segmentation ground-truth, the results are sufficiently different to support a qualitative ranking of the different approaches. Figure IV.9 illustrates the performance on the first sequence, which depicts a water fountain with three regions of different dynamics: water flowing down a wall, falling water, and turbulent water in a pool. While DytxMixCS separates the three regions, GPCA does not produce a sensible segmentation. This result confirms the observations obtained with the synthetic database of the previous section.

We next consider macroscopic dynamic textures, in which case segmentation performance is compared against that of traditional motion-based solutions, i.e. the combination of normalized-cuts with motion profiles (NormCuts), or optical flow with mean-shift clustering (OpFlow). Figure IV.10a shows a scene of highway traffic, for which DytxMixCS is the only method that correctly segments

---

4The other methods could not be applied to this sequence because it contains three regions.
the video into regions of traffic that move away from the camera (the two large regions on the right) and traffic that move towards the camera (the regions on the left). The only error is the split of the incoming traffic into two regions, and can be explained by the strong perspective effects inherent to car motion towards the camera. This could be avoided by applying an inverse perspective mapping to the scene, but we have not considered any such geometric transformations. Figure IV.10b shows another example of segmentation of vehicle traffic on a bridge. Traffic lanes of opposite direction are correctly segmented near the camera, but merged further down the bridge. The algorithm also segments the water in the bottom-right of the image, but assigns it to the same cluster as the distant traffic. While not perfect, these segmentations are significantly better than those produced by the traditional representations. For both NormCuts and OpFlow, segmented regions tend to extend over multiple lanes, incoming and outgoing traffic are merged, and the same lane is frequently broken into a number of sub-regions.

The final two videos are of pedestrian scenes. The first scene, shown in Figure IV.11a, contains sparse pedestrian traffic, i.e. with large gaps between pedestrians. DytexMixCS (Figure IV.11b) segments people moving up the walkway from people moving down the walkway. The second scene (Figure IV.11c) contains a large crowd moving up the walkway, with only a few people moving in the opposite direction. Again, DytexMixCS (Figure IV.11d) segmented the groups moving in different directions, even in instances where only one person is surrounded by the crowd and moving in the opposite direction. Figure IV.11e and Figure IV.11f show the segmentations produced by the traditional methods. The
Figure IV.10 Segmentation of traffic scenes: (top) highway traffic; (bottom) vehicle traffic on bridge; The left column shows a frame from the original videos, while the remaining columns show the segmentations.

<table>
<thead>
<tr>
<th></th>
<th>original</th>
<th>DytexMixCS</th>
<th>NormCuts</th>
<th>OpFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td><img src="image1.jpg" alt="Frame" /></td>
<td><img src="image2.jpg" alt="Segmentation" /></td>
<td><img src="image3.jpg" alt="Segmentation" /></td>
<td><img src="image4.jpg" alt="Segmentation" /></td>
</tr>
<tr>
<td>(b)</td>
<td><img src="image5.jpg" alt="Frame" /></td>
<td><img src="image6.jpg" alt="Segmentation" /></td>
<td><img src="image7.jpg" alt="Segmentation" /></td>
<td><img src="image8.jpg" alt="Segmentation" /></td>
</tr>
</tbody>
</table>

Figure IV.11 Segmentation of two pedestrian scenes: (a) pedestrian scene with sparse traffic, and (b) the segmentation by DytexMixCS; (c) Pedestrian scene with heavy traffic, and the segmentations by (d) DytexMixCS; (e) NormCuts, and (f) OpFlow.

<table>
<thead>
<tr>
<th></th>
<th>original</th>
<th>DytexMixCS</th>
<th>NormCuts</th>
<th>OpFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td><img src="image9.jpg" alt="Frame" /></td>
<td><img src="image10.jpg" alt="Segmentation" /></td>
<td><img src="image11.jpg" alt="Segmentation" /></td>
<td><img src="image12.jpg" alt="Segmentation" /></td>
</tr>
<tr>
<td>(b)</td>
<td><img src="image13.jpg" alt="Frame" /></td>
<td><img src="image14.jpg" alt="Segmentation" /></td>
<td><img src="image15.jpg" alt="Segmentation" /></td>
<td><img src="image16.jpg" alt="Segmentation" /></td>
</tr>
<tr>
<td>(c)</td>
<td><img src="image17.jpg" alt="Frame" /></td>
<td><img src="image18.jpg" alt="Segmentation" /></td>
<td><img src="image19.jpg" alt="Segmentation" /></td>
<td><img src="image20.jpg" alt="Segmentation" /></td>
</tr>
<tr>
<td>(d)</td>
<td><img src="image21.jpg" alt="Frame" /></td>
<td><img src="image22.jpg" alt="Segmentation" /></td>
<td><img src="image23.jpg" alt="Segmentation" /></td>
<td><img src="image24.jpg" alt="Segmentation" /></td>
</tr>
<tr>
<td>(e)</td>
<td><img src="image25.jpg" alt="Frame" /></td>
<td><img src="image26.jpg" alt="Segmentation" /></td>
<td><img src="image27.jpg" alt="Segmentation" /></td>
<td><img src="image28.jpg" alt="Segmentation" /></td>
</tr>
<tr>
<td>(f)</td>
<td><img src="image29.jpg" alt="Frame" /></td>
<td><img src="image30.jpg" alt="Segmentation" /></td>
<td><img src="image31.jpg" alt="Segmentation" /></td>
<td><img src="image32.jpg" alt="Segmentation" /></td>
</tr>
</tbody>
</table>
segmentation produced by NormCuts contains gross errors, e.g. frame 2 at the far end of the walkway. While the segmentation achieved with OpFlow is more comparable to that of DytexMixCS, it tends to over-segment the people moving down the walkway.

Finally, we illustrate the point that the mixture of dynamic textures can be used to efficiently segment very long video sequences. Using the procedure discussed in Section IV.E.2, a continuous hour of pedestrian video was segmented with the mixture model learned from Figure IV.11c, and is available for visualization in the supplemental material [52]. It should be noted that this segmentation required no reinitialization at any point, or any other type of manual supervision. We consider this a significant result, given that this sequence contains a fair variability of traffic density, various outlying events (e.g. bicycles, skateboarders, or even small vehicles that pass through, pedestrians that change course or stop to chat, etc.), and variable environmental conditions (such as varying clouds shadows). None of these factors appear to influence the performance of the DytexMixCS algorithm. To the best of our knowledge, this is the first time that a coherent video segmentation, over a time span of this scale, has been reported for a crowded scene.

IV.G Summary and discussion

In this chapter, we have studied the mixture of dynamic textures, a principled probabilistic extension of the dynamic texture model. Whereas a dynamic texture models a single video sequence as a sample from a linear dynamic system, a mixture of dynamic textures models a collection of sequences as samples from a set of linear dynamic systems. We derived an exact EM algorithm for learning the parameters of the model from a set of training video, and explored the connections between the model and other linear system models, such as factor analysis, mixtures of factor analyzers, and switching linear systems.

Through extensive video clustering and segmentation experiments, we
have also demonstrated the efficacy of the mixture of dynamic textures for modeling video, both holistically and locally (patch-based representations). In particular, it has been shown that the mixture of dynamic textures is a suitable model for simultaneously representing the localized motion and appearance of a variety of visual processes (e.g. fire, water, steam, cars, and people), and that the model provides a natural framework for clustering such processes. In the application of motion segmentation, the experimental results indicate that the mixture of dynamic textures provides better segmentations than other state-of-the-art methods, based on either dynamic textures or on traditional representations. Some of the results, namely the segmentation of pedestrian scenes, suggest that the dynamic texture mixture could be the basis for the design of computer vision systems capable of tackling problems, such as the monitoring and surveillance of crowded environments, which currently have great societal interest. In particular, in Chapter VIII we utilize the DTM motion segmentation algorithm in a surveillance system for estimating the size of inhomogeneous crowds, composed of pedestrians that travel in different directions.

There are also some issues which we leave open for future work. One example is how to incorporate, in the dynamic texture mixture framework, some recent developments in asymptotically efficient estimators based on non-iterative sub-space methods [30]. By using such estimators in the M-step of the EM algorithm, it may be possible to reduce the number of hidden variables required in the E-step, and consequently improve the convergence properties of EM. It is currently unclear if the optimality of these estimators is compromised when the initial state has arbitrary covariance, or when the LDS is learned from multiple sample paths, as is the case for dynamic texture mixtures.

Another open question is that of the identifiability of a LDS, when the initial state has arbitrary covariance. It is well known, in the system identification literature, that the parameters of a LDS can only be identified from a single spatio-temporal sample if the covariance of the initial condition satisfies a Lya-
punov condition. In the absence of identifiability, the solution may not be unique, may not exist, learning may not converge, or the estimates may not be consistent. It is important to note that none of these problems are of great concern for the methods discussed in this chapter. For example, it is well known that EM is guaranteed to converge to a local maximum of the likelihood function, and produces asymptotically consistent parameter estimates. These properties are not contingent on the identifiability of the LDS components. Although the local maxima of the likelihood could be ridges (i.e. not limited to points but supported by manifolds), in which case the optimal component parameters would not be unique, there would be no consequence for segmentation or clustering as long as all computations are based on likelihoods (or other probabilistic distance measures such as the Kullback-Leibler divergence). Non-uniqueness could, nevertheless, be problematic for procedures that rely on direct comparison of the component parameters (e.g. based on their Euclidean distances), which we do not advise. In any case, it would be interesting to investigate more thoroughly the identifiability question. The fact that we have not experienced convergence speed problems, in the extensive experiments discussed above, indicates that likelihood ridges are unlikely. In future work, we will attempt to understand this question more formally.

IV.H Acknowledgements

The authors thank Gianfranco Doretto and Stefano Soatto for the synthetic sequences from [15, 16], the Washington State DOT [53] for the videos of highway traffic, Daniel Dailey for the loop-sensor data, Rene Vidal for code from [19, 17], Pedro Moreno for helpful discussions.

The text of Chapter IV, in full, is based on the material as it appears in: A. B. Chan and N. Vasconcelos, “Modeling, clustering, and segmenting video with mixtures of dynamic textures”, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 30(5), May 2008. The mixture of dynamic textures was originally

IV.I Appendix: EM algorithm for the mixture of dynamic textures

This appendix presents the derivation of the EM algorithm for the mixture of dynamic textures. In particular, the complete-data log-likelihood function, the E-step, and the M-step are derived in the remainder of this appendix.

Log-Likelihood Functions

We start by obtaining the log-likelihood of the complete-data (see Table IV.1 for notation). Using (IV.6) and the indicator variables $z_{i,j}$, the complete-data log-likelihood is

$$\ell(X, Y, Z) = \sum_{i=1}^{N} \log p(x^{(i)}, y^{(i)}, z^{(i)})$$

$$= \sum_{i=1}^{N} \log \prod_{j=1}^{K} [p(x^{(i)}, y^{(i)}, z^{(i)} = j)]^{z_{i,j}}$$

$$= \sum_{i,j} z_{i,j} \log \left[ \alpha_j p(x_1^{(i)}|z^{(i)} = j) \prod_{t=2}^{\tau} p(x_t^{(i)}|x_{t-1}^{(i)}, z^{(i)} = j) \prod_{t=1}^{\tau} p(y_t^{(i)}|x_t^{(i)}, z^{(i)} = j) \right]$$

$$= \sum_{i,j} z_{i,j} \left[ \log \alpha_j + \log p(x_1^{(i)}|z^{(i)} = j) + \sum_{t=2}^{\tau} \log p(x_t^{(i)}|x_{t-1}^{(i)}, z^{(i)} = j) \right.$$

$$+ \sum_{t=1}^{\tau} \log p(y_t^{(i)}|x_t^{(i)}, z^{(i)} = j) \left. \right].$$
Note that, from (IV.3)-(IV.5), the sums of the log-conditional probability terms are of the form

\[
\sum_{i,j} a_{i,j} \sum_{t=t_0}^{t_1} \log G(b_t, c_{j,t}, M_j) = -\frac{n}{2}(t_1 - t_0 + 1) \log 2\pi \sum_{i,j} a_{i,j} \quad \text{(IV.29)}
\]

\[
- \frac{1}{2} \sum_{i,j} a_{i,j} \sum_{t=t_0}^{t_1} ||b_t - c_{j,t}||^2_{M_j} - \frac{1}{2} \sum_{i,j} \sum_{t=t_0}^{t_1} a_{i,j} \log |M_j|.
\]

Since the first term on the right-hand side of this equation does not depend on the parameters of the dynamic texture mixture, it does not affect the maximization performed in the M-step and can, therefore, be dropped. Substituting the appropriate parameters for \(b_t, c_{j,t}\) and \(M_j\), we have:

\[
\ell(X, Y, Z) = \sum_{i,j} \alpha_j \quad \text{(IV.30)}
\]

\[
- \frac{1}{2} \sum_{i,j} \log |S_j| - \frac{1}{2} \sum_{i,j} \left\| x_1^{(i)} - \mu_j \right\|_{S_j}^2
\]

\[
- \frac{\tau}{2} \sum_{i,j} \log |Q_j| - \frac{1}{2} \sum_{i,j} \tau \sum_{t=2}^{\tau} \left\| x_t^{(i)} - A_j x_{t-1}^{(i)} \right\|_{Q_j}^2
\]

\[
- \frac{\tau}{2} \sum_{i,j} \log |R_j| - \frac{1}{2} \sum_{i,j} \tau \sum_{t=1}^{\tau} \left\| y_t^{(i)} - C_j x_t^{(i)} \right\|_{R_j}^2.
\]

Defining the random variables \(P_{t,t}^{(i)} = x_t^{(i)}(x_t^{(i)})^T\) and \(P_{t,t-1}^{(i)} = x_t^{(i)}(x_{t-1}^{(i)})^T\) and expanding the Mahalanobis distance terms, the log-likelihood becomes (IV.10).

**E-Step**

The E-step of the EM algorithm is to take the expectation of (IV.10) conditioned on the observed data and the current parameter estimates \(\hat{\Theta}\), as in (IV.8). We note that each term of \(\ell(X, Y, Z)\) is of the form \(z_{i,j} f(x^{(i)}, y^{(i)})\), for some functions \(f\) of \(x^{(i)}\) and \(y^{(i)}\), and its expectation is

\[
E_{X,Z|Y} \left( z_{i,j} f(x^{(i)}, y^{(i)}) \right) \quad \text{(IV.31)}
\]

\[
= E_{Z|Y} \left( E_{X|Y,Z} \left( z_{i,j} f(x^{(i)}, y^{(i)}) \right) \right) \quad \text{(IV.32)}
\]

\[
= E_{z^{(i)}|y^{(i)}} \left( E_{x^{(i)}|y^{(i),z^{(i)}}}(z_{i,j} f(x^{(i)}, y^{(i)})) \right) \quad \text{(IV.33)}
\]

\[
= p(z_{i,j} = 1|y^{(i)}) E_{x^{(i)}|y^{(i),z^{(i)}} = j}( f(x^{(i)}, y^{(i)}) ), \quad \text{(IV.34)}
\]
where (IV.33) follows from the assumption that the observations are independent. For the first term of (IV.34), \( p(z_{i,j} = 1 | y^{(i)}) \) is the posterior probability of \( z^{(i)} = j \) given the observation \( y^{(i)} \),

\[
\hat{z}_{i,j} = p(z_{i,j} = 1 | y^{(i)}) = p(z^{(i)} = j | y^{(i)})
\]

(IV.35)

\[
\hat{z}_{i,j} = \frac{\alpha_j p(y^{(i)} | z^{(i)} = j)}{\sum_{k=1}^{K} \alpha_k p(y^{(i)} | z^{(i)} = k)}.
\]

(IV.36)

The functions \( f(x^{(i)}, y^{(i)}) \) are at most quadratic in \( x_t^{(i)} \). Hence, the second term of (IV.34) only depends on the first and second moments of the states conditioned on \( y^{(i)} \) and component \( j \) (IV.13-IV.15), and are computed as described in Section IV.C.1. Finally, the \( Q \) function (IV.11) is obtained by first replacing the random variables \( z_{i,j}^{(i)}, (z_{i,j} x_t^{(i)}, (z_{i,j} P_{t,t}^{(i)}), \) and \( (z_{i,j} P_{t,t-1}^{(i)}) \) in the complete-data log-likelihood (IV.10) with the corresponding expectations \( \hat{z}_{i,j}, (\hat{z}_{i,j} \hat{x}_{t,t}^{(i)}, (\hat{z}_{i,j} \hat{P}_{t,t}^{(i)}), \) and \( (\hat{z}_{i,j} \hat{P}_{t,t-1}^{(i)}) \), and then defining the aggregated expectations (IV.12).

M-Step

In the M-step of the EM algorithm (IV.9), the reparameterization of the model is obtained by maximizing the \( Q \) function by taking the partial derivative with respect to each parameter and setting it to zero. The maximization problem with respect to each parameter appears in two common forms. The first is a maximization with respect to a square matrix \( X \)

\[
X^* = \arg\max_X -\frac{1}{2} \text{tr} \left( X^{-1} A \right) - b \frac{1}{2} \log |X|.
\]

(IV.37)

Maximizing by taking the derivative and setting to zero yields the following solution

\[
\frac{\partial}{\partial X} -\frac{1}{2} \text{tr} \left( X^{-1} A \right) - b \frac{1}{2} \log |X| = 0
\]

(IV.38)

\[
\frac{1}{2} X^{-T} A^T X^{-T} - b X^{-T} = 0
\]

(IV.39)

\[
A^T - b X^T = 0,
\]

(IV.40)

\[
\Rightarrow X^* = \frac{1}{b} A.
\]

(IV.41)
The second form is a maximization problem with respect to a matrix $X$ of the form

$$X^* = \operatorname{argmax}_X -\frac{1}{2} \operatorname{tr} \left[ D(-BX^T - XB^T + XCX^T) \right],$$  

(IV.42)

where $D$ and $C$ are symmetric and invertible matrices. The maximum is given by

$$\frac{\partial}{\partial X} -\frac{1}{2} \operatorname{tr} \left[ D(-BX^T - XB^T + XCX^T) \right] = 0$$  

(IV.43)

$$-\frac{1}{2} (-DB - D^TB + D^TXC^T + DXC) = 0$$  

(IV.44)

$$DB - DXC = 0,$$  

(IV.45)

$$\Rightarrow X^* = BC^{-1}.$$  

(IV.46)

The optimal parameters are found by collecting the relevant terms in (IV.11) and maximizing:

1. **Observation Matrix:**

   $$C^*_j = \operatorname{argmax}_{C_j} -\frac{1}{2} \operatorname{tr} \left[ R_j^{-1}(-\Gamma_j C_j^T - C_j \Gamma_j^T + C_j \Phi_j C_j^T) \right].$$

   This is of the form in (IV.42), hence the solution is given by $C^*_j = \Gamma_j (\Phi_j)^{-1}$.

2. **Observation Noise Covariance:**

   $$R^*_j = \operatorname{argmax}_{R_j} -\frac{\tau \hat{N}_j}{2} \log |R_j|$$

   $$-\frac{1}{2} \operatorname{tr} \left[ R_j^{-1}(-\Lambda_j + \Gamma_j C_j^T - C_j \Gamma_j^T + C_j \Phi_j C_j^T) \right].$$

   This is of the form in (IV.37), hence the solution is

   $$R^*_j = \frac{1}{\tau \hat{N}_j} \left( \Lambda_j - \Gamma_j C_j^T - C_j \Gamma_j^T + C_j \Phi_j C_j^T \right)$$  

   (IV.48)

   $$= \frac{1}{\tau \hat{N}_j} \left( \Lambda_j - C_j^* \Gamma_j^T \right),$$  

   (IV.49)

where (IV.49) follows from substituting for the optimal value $C_j^*$. 
3. State Transition Matrix:

\[ A_j^* = \arg\max_{A_j} -\frac{1}{2} \text{tr}\left[ Q_j^{-1} \left( -\Psi_j A_j^T - A_j \Psi_j^T + A_j \phi_j A_j^T \right) \right]. \]

This is of the form in (IV.42), hence \( A_j^* = \Psi_j (\phi_j)^{-1}. \)

4. State Noise Covariance:

\[ Q_j^* = \arg\max_{Q_j} -\frac{(\tau - 1)\hat{N}_j}{2} \log |Q_j| \]

\[ -\frac{1}{2} \text{tr}\left[ Q_j^{-1} \left( \varphi_j - \Psi_j A_j^T - A_j \Psi_j^T + A_j \phi_j A_j^T \right) \right]. \]

This is of the form in (IV.37), hence the solution can be computed as

\[ Q_j^* = \frac{1}{(\tau - 1)\hat{N}_j} \left( \varphi_j - A_j^* \Psi_j^T \right), \]

where (IV.51) follows from substituting for the optimal value \( A_j^*. \)

5. Initial State Mean:

\[ \mu_j^* = \arg\max_{\mu_j} -\frac{1}{2} \text{tr}\left[ S_j^{-1} \left( -\xi_j \mu_j^T - \mu_j \xi_j^T + \hat{N}_j \mu_j \mu_j^T \right) \right]. \]

This is of the form in (IV.42), hence the solution is given by \( \mu_j^* = \frac{1}{\hat{N}_j} \xi_j. \)

6. Initial State Covariance:

\[ S_j^* = \arg\max_{S_j} -\frac{\hat{N}_j}{2} \log |S_j| \]

\[ -\frac{1}{2} \text{tr}\left[ S_j^{-1} \left( \eta_j - \xi_j \mu_j^T - \mu_j \xi_j^T + \hat{N}_j \mu_j \mu_j^T \right) \right]. \]

This is of the form in (IV.37), hence the solution is given by

\[ S_j^* = \frac{1}{\hat{N}_j} \left( \eta_j - \xi_j \mu_j^* - \mu_j \xi_j^* + \hat{N}_j \mu_j \mu_j^* \right) \]

\[ S_j^* = \frac{1}{\hat{N}_j} \eta_j - \mu_j^*(\mu_j^*)^T, \]

where (IV.54) follows from substituting for the optimal value \( \mu_j^*. \)
7. *Class Probabilities:* A Lagrangian multiplier is used to enforce that \( \{\alpha_j\} \) sum to 1,

\[
\alpha = \arg \max_{\alpha, \lambda} \sum_j \hat{N}_j \log \alpha_j + \lambda \left( \sum_j \alpha_j - 1 \right),
\]  

(IV.55)

where \( \alpha = \{\alpha_1, \cdots, \alpha_K\} \). The optimal value is \( \alpha_j^* = \frac{\hat{N}_j}{N} \).
Chapter V

Layered dynamic textures
V.A Introduction

Traditional motion representations, based on optical flow, are inherently local and have significant difficulties when faced with aperture problems and noise. The classical solution to this problem is to regularize the optical flow field [1, 2, 3, 4], but this introduces undesirable smoothing across motion edges or regions where the motion is, by definition, not smooth (e.g. vegetation in outdoors scenes). It also does not provide any information about the objects that compose the scene, although the optical flow field could be subsequently used for motion segmentation.

More recently, there have been various attempts to model video as a superposition of layers subject to homogeneous motion. While layered representations exhibited significant promise in terms of combining the advantages of regularization (use of global cues to determine local motion) with the flexibility of local representations (little undue smoothing), and a truly object-based representation, this potential has so far not fully materialized. One of the main limitations is their dependence on parametric motion models, such as affine transforms, which assume a piece-wise planar world that rarely holds in practice [6, 79]. In fact, layers are usually formulated as “cardboard” models of the world that are warped by such transformations and then stitched to form the frames in a video stream [6]. This severely limits the types of video that can be synthesized: while the concept of layering showed most promise for the representation of scenes composed of ensembles of objects subject to homogeneous motion (e.g. leaves blowing in the wind, a flock of birds, a picket fence, or highway traffic), very little progress has so far been demonstrated in actually modeling such scenes.

Recently, there has been more success in modeling complex scenes as dynamic textures or, more precisely, samples from stochastic processes defined over space and time [13, 15, 16]. This work has demonstrated that global stochastic modeling of both video dynamics and appearance is much more powerful than the classic global modeling as “cardboard” figures under parametric motion. In fact,
the dynamic texture (DT) has shown a surprising ability to abstract a wide variety of complex patterns of motion and appearance into a simple spatio-temporal model. One major current limitation is, however, its inability to account for visual processes consisting of multiple, co-occurring, dynamic textures, for example, a flock of birds flying in front of a water fountain, highway traffic moving at different speeds, and video containing both trees in the background and people in the foreground. In such cases, the existing DT model is inherently ill-equipped to model the video, since it must represent multiple motion fields with a single dynamic process.

To address this problem, various extensions of the DT have been recently proposed in the literature [15, 17, 19]. These extensions have emphasized the application of the standard DT model to video segmentation, rather than exploiting the probabilistic nature of the DT representation to propose a global generative model for video. They represent the video as a collection of localized spatio-temporal patches (or pixel trajectories). These are then modeled with dynamic textures, or similar time-series representations, whose parameters are clustered to produce the desired segmentations. Due to their local character, these representations cannot account for globally homogeneous textures that exhibit substantial local heterogeneity. These types of textures are common in both urban settings, where the video dynamics frequently combine global motion and stochasticity (e.g. vehicle traffic around a square, or pedestrian traffic around a landmark), and natural scenes (e.g. a flame that tilts under the influence of the wind, or water rotating in a whirlpool).

In this chapter, we address this limitation by introducing a new generative model for video, which we denote by the layered dynamic texture (LDT). This consists of augmenting the dynamic texture with a discrete hidden variable, that enables the assignment of different dynamics to different regions of the video. The hidden variable is modeled as a Markov random field (MRF) to ensure spatial smoothness of the regions, and conditioned on the state of this hidden variable, each region of video is a standard DT. By introducing a shared dynamic represen-
tation for all pixels in a region, the new model is a layered representation. When compared with traditional layered models, it replaces layer formation by “warping cardboard figures” with sampling from the generative model (for both dynamics and appearance) provided by the DT. This enables a much richer video representation. Since each layer is a DT, the model can also be seen as a multi-state dynamic texture, which is capable of assigning different dynamics and appearance to different image regions. In addition to introducing the model, we derive the EM algorithm for maximum-likelihood estimation of its parameters from an observed video sample. Because exact inference is computationally intractable (due to the MRF), we present two approximate inference algorithms: a Gibbs sampler and a variational approximation. Finally, we apply the LDT to motion segmentation of challenging video sequences.

The remainder of the chapter is organized as follows. In Section V.B, we introduce the LDT model. In Section V.C we derive the EM algorithm for parameter learning. Sections V.D and V.E then propose the Gibbs sampler and variational approximation. In Section V.F, we propose a temporally-switching LDT that changes layer assignments at each frame. Finally, in Section V.G we present an experimental evaluation of the two approximate inference algorithms, and in Section V.H apply the LDT to motion segmentation of both synthetic and real video sequences.

V.B Layered dynamic textures

In this chapter we consider video composed of various textures, e.g. the combination of fire, smoke, and water shown on the right side of Figure V.1. In this case, a single DT cannot simultaneously account for the appearance and dynamics of the three textures, because each texture moves distinctly. For example, fire changes faster and is more chaotic than water or smoke. As shown in Figure V.1, this type of video can be modeled by encoding each texture as a separate
Figure V.1 Generative model for a video with multiple dynamic textures (smoke, water, and fire). The three textures are modeled with separate state sequences and observation matrices. The textures are then masked, and compositing to form the layered video.

layer, with its own state-sequence and observation matrix. Different regions of the spatiotemporal video volume are assigned to each texture and, conditioned on this assignment, each region evolves as a standard DT. The video is a composite of the various layers.

Formally, the graphical model for the layered dynamic texture is shown in Figure V.2a. Each of the $K$ layers has a state process $x^{(j)} = \{x_t^{(j)}\}$ that evolves separately. A pixel trajectory $y_i = \{y_{i,t}\}$ is assigned to one of the layers through the hidden variable $z_i$. The collection of hidden variables $Z = \{z_i\}_{i=1}^m$ is modeled as a Markov random field (MRF) to ensure spatial smoothness of the layer assignments (e.g. Figure V.2b). The model equations are

$$
\begin{cases}
x_t^{(j)} = A^{(j)} x_{t-1}^{(j)} + v_t^{(j)}, & j \in \{1, \cdots, K\} \\
y_{i,t} = C_i^{(z_i)} x_t^{(z_i)} + w_{i,t}, & i \in \{1, \cdots, m\}
\end{cases}
$$

(V.1)

where $C_i^{(j)} \in \mathbb{R}^{1 \times n}$ is the transformation from the hidden state to the observed pixel for each pixel $y_i$ and each layer $j$. The noise processes are $v_t^{(j)} \sim \mathcal{N}(0, Q^{(j)})$ and $w_{i,t} \sim \mathcal{N}(0, r^{(z_i)})$, and the initial state is given by $x_1^{(j)} \sim \mathcal{N}(\mu^{(j)}, Q^{(j)})$, where $Q^{(j)} \in \mathbb{S}_+^{n}$ and $r^{(j)} \in \mathbb{R}_+$. Note that for the LDT we assume the initial state has covariance
Figure V.2  a) Graphical model for the LDT. $y_i$ is an observed pixel process and $x^{(j)}$ a hidden state process. $z_i$ assigns $y_i$ to one of the state processes, and the collection $\{z_i\}$ is modeled as an MRF; (b) Example of a $4 \times 4$ layer assignment MRF.

$$S^{(j)} = Q^{(j)}$$ for identifiability purposes. Each layer is parameterized by $\Theta_j = \{A^{(j)}, Q^{(j)}, C^{(j)}, r^{(j)}, \mu^{(j)}\}$. Given layer assignments, the LDT is a superposition of DTs defined over different regions of the video volume, and estimating the parameters of the LDT reduces to estimating those of the DT of each region. When layer assignments are unknown, model parameters can be estimated with the EM algorithm (see Section V.C). We next derive the joint probability distribution of the LDT.

V.B.1 Joint distribution of the LDT

As is typical for mixture models, we introduce an indicator variable $z^{(j)}_i$ of value 1 if and only if $z_i = j$, and 0 otherwise. The LDT model assumes that the state processes $X = \{x^{(j)}\}_{j=1}^K$ and the layer assignments $Z$ are independent, i.e. the layer dynamics are independent of its location. Under this assumption, the joint distribution factors as

$$p(X, Y, Z) = p(Y|X, Z)p(X)p(Z) \quad (V.2)$$

$$= \prod_{i=1}^m \prod_{j=1}^K p(y_i|x^{(j)}, z_i = j)^{z^{(j)}_i} \prod_{j=1}^K p(x^{(j)})p(Z), \quad (V.3)$$
where $Y = \{y_i\}_{i=1}^m$. Each state-sequence is a Gauss-Markov process, with distribution

$$p(x^{(j)}) = p(x_1^{(j)}) \prod_{t=2}^\tau p(x_t^{(j)}|x_{t-1}^{(j)}),$$

where the individual state densities are

$$p(x_1^{(j)}) = G(x_1^{(j)}, \mu^{(j)}, Q^{(j)}), \quad p(x_t^{(j)}|x_{t-1}^{(j)}) = G(x_t^{(j)}, A^{(j)}x_{t-1}^{(j)}, Q^{(j)}),$$

and $G(x, \mu, \Sigma)$ is a Gaussian of mean $\mu$ and covariance $\Sigma$. When conditioned on state sequences and layer assignments, pixel values are independent, and pixel trajectories distributed as

$$p(y_i|x^{(j)}, z_i = j) = \prod_{t=1}^\tau p(y_{i,t}|x_t^{(j)}, z_i = j),$$

where

$$p(y_{i,t}|x_t^{(j)}, z_i = j) = G(y_{i,t}, C^{(j)}_i x_t^{(j)}, r^{(j)}).$$

Finally, the layer assignments are jointly distributed as

$$p(Z) = \frac{1}{Z_Z} \prod_{i=1}^m V_i(z_i) \prod_{(i,i') \in \mathcal{E}} V_{i,i'}(z_i, z_{i'}),$$

where $\mathcal{E}$ is the set of edges of the MRF, $Z_Z$ a normalization constant (partition function), and $V_i$ and $V_{i,i'}$ potential functions of the form

$$V_i(z_i) = \prod_{j=1}^K (\alpha_i^{(j)})_{z_i = j}^{(j)} = \begin{cases} \alpha_i^{(1)}, z_i = 1 \\ \vdots \\ \alpha_i^{(K)}, z_i = K \end{cases},$$

$$V_{i,i'}(z_i, z_{i'}) = \gamma_2 \prod_{j=1}^K \left( \frac{\gamma_1}{\gamma_2} \right)_{z_i = j}^{(j)} z_i^{(j)} z_{i'}^{(j)} = \begin{cases} \gamma_1, z_i = z_{i'} \\ \gamma_2, z_i \neq z_{i'} \end{cases}.$$

$V_i$ is the prior probability of each layer, while $V_{i,i'}$ attributes higher probability to configurations with neighboring pixels in the same layer. In this work, we treat the MRF as a prior on $Z$, which controls the smoothness of the layers. The parameters of the potential functions of each layer could be learned, in a manner similar to [86], but we have so far found this to be unnecessary.
V.B.2 Related work

A number of applications of DT (or similar) models to segmentation have been reported in the literature. [15] models spatio-temporal patches extracted from video as DTs, and clusters them using level-sets and the Martin distance. [19] clusters pixel-intensities (or local texture features) using auto-regressive processes (AR) and level-sets. [17] segments video by clustering pixels with similar trajectories in time using generalized PCA (GPCA), while [87] clusters pixel trajectories lying in multiple moving hyperplanes using an on-line recursive algorithm to estimate polynomial coefficients. While these methods have shown promise, they do not exploit the probabilistic nature of the DT representation for the segmentation itself. A different approach is proposed by [88], which segments high-density crowds with a Lagrangian particle dynamics model, where the flow field generated by a moving crowd is treated as an aperiodic dynamical system. Although promising, this work is limited to scenes where the optical flow can be reliably estimated (e.g. crowds of moving people, but not moving water).

More related to the extensions proposed in this chapter is the mixture of dynamic textures from Chapter IV. This is a model for collections of video sequences, and has been successfully used for motion-based video segmentation through clustering of spatio-temporal patches. The main difference with respect to the LDT now proposed is that (like all clustering models) the mixture of DTs is not a global generative model for video of co-occurring textures (as is the case of the LDT). Hence, the application of the mixture of DTs to segmentation requires decomposing the video into a collection of small spatio-temporal patches, which are then clustered. The localized nature of this video representation is problematic for the segmentation of textures which are globally homogeneous but exhibit substantial variation between neighboring locations, such as the rotating motion of the water in a whirlpool. Furthermore, patch-based segmentations have poor boundary accuracy, due to the artificial boundaries of the underlying patches, and the difficulty of assigning a patch that overlaps multiple regions to any of them.
On the other hand, the LDT models video as a collection of layers, offering a truly global model of the appearance and dynamics of each layer, and avoiding boundary uncertainty. With respect to time-series models, the LDT is related to switching linear dynamical models, which are LDSs that can switch between different parameter sets over time \[62, 63, 64, 65, 66, 67, 89, 69\]. In particular, it is most related to the switching state-space LDS \[69\], which models the observed variable by switching between the outputs of a set of independent LDSs. The fundamental difference between the two models is that, while \[69\] switches parameters in time using a hidden-Markov model (HMM), the LDT switches parameters in space (i.e. within the dimensions of the observed variable) using an MRF. This substantially complicates all statistical inference, leading to very different algorithms for learning and inference with LDTs.

\textbf{V.C Parameter estimation with the EM algorithm}

Given a training video \(Y\), the parameters \(\Theta\) are learned by maximum-likelihood\[24]\n
\[
\Theta^* = \arg\max_{\Theta} \log p(Y) = \arg\max_{\Theta} \log \sum_{X, Z} p(Y, X, Z). \tag{V.10}
\]

Since the data likelihood depends on hidden variables (state sequence \(X\) and layer assignments \(Z\)) this problem can be solved with the EM algorithm \[31\], which iterates between

\[
\text{E - Step : } Q(\Theta; \hat{\Theta}) = \mathbb{E}_{X, Z | Y, \hat{\Theta}}[\ell(X, Y, Z; \Theta)], \tag{V.11}
\]

\[
\text{M - Step : } \hat{\Theta}' = \arg\max_{\Theta} Q(\Theta; \hat{\Theta}), \tag{V.12}
\]

where \(\ell(X, Y, Z; \Theta) = \log p(X, Y, Z; \Theta)\) is the complete-data log-likelihood, parameterized by \(\Theta\), and \(\mathbb{E}_{X, Z | Y, \hat{\Theta}}\) the expectation with respect to \(X\) and \(Z\), conditioned on \(Y\), parameterized by the current estimates \(\hat{\Theta}\). In the remainder of this section we derive the E and M steps for the LDT model. We will assume, without loss
of generality, that the video $Y$ has zero-mean. If not, the mean can simply be subtracted before learning, and subsequently added to each layer.

V.C.1 Complete data log-likelihood

Taking the logarithm of (V.3), the complete data log-likelihood is

$$
\ell(X, Y, Z) = \sum_{i=1}^{m} \sum_{j=1}^{K} \sum_{t=1}^{\tau} \log p(y_{i,t} | x_{t}^{(j)}, z_{i} = j) - \frac{1}{2} \sum_{j=1}^{K} \sum_{i=1}^{m} z_{i}^{(j)} \sum_{t=1}^{\tau} \left( \| y_{i,t} - C_{i}^{(j)} x_{t}^{(j)} \|_{r_j(j)}^{2} + \log r(j) \right) - \frac{1}{2} \sum_{j=1}^{K} \sum_{i=1}^{m} \sum_{t=1}^{\tau} \left( \| x_{t}^{(j)} - \mu^{(j)} \|_{Q(j)}^{2} + \sum_{t=2}^{\tau} \| x_{t}^{(j)} - A^{(j)} x_{t-1}^{(j)} \|_{Q(j)}^{2} + \tau \log |Q(j)| \right),
$$

where $\|x\|_{\Sigma}^{2} = x^{T} \Sigma^{-1} x$. Note that $p(Z)$ can be ignored since the parameters of the MRF are constants. Finally, the complete data log-likelihood is

$$
\ell(X, Y, Z) = -\frac{1}{2} \sum_{j=1}^{K} \sum_{i=1}^{m} z_{i}^{(j)} \sum_{t=1}^{\tau} \frac{1}{r(j)} \left( y_{i,t}^{2} - 2y_{i,t} C_{i}^{(j)} x_{t}^{(j)} + C_{i}^{(j)} P_{t,t}^{(j)} C_{i}^{(j)T} \right) - \frac{1}{2} \sum_{j=1}^{K} \text{tr} \left( Q(j)^{-1} \left( P_{1,1}^{(j)} - x_{1}^{(j)} \mu^{(j)} T - \mu^{(j)} x_{1}^{(j)} T + \mu^{(j)} \mu^{(j)} T \right) \right) - \frac{1}{2} \sum_{j=1}^{K} \sum_{t=2}^{\tau} \text{tr} \left( Q(j)^{-1} \left( P_{t,t}^{(j)} - P_{t,t-1} A^{(j)} T - A^{(j)} P_{t,t-1}^{(j)T} + A^{(j)} P_{t,t-1,t-1} A^{(j)T} \right) \right) - \frac{\tau}{2} \sum_{j=1}^{K} \sum_{i=1}^{m} z_{i}^{(j)} \log r(j) - \frac{\tau}{2} \sum_{j=1}^{K} \log |Q(j)|,
$$

where we define $P_{t,t}^{(j)} = x_{t}^{(j)} x_{t}^{(j)T}$ and $P_{t,t-1}^{(j)} = x_{t}^{(j)} x_{t-1}^{(j)T}$. 

The log-likelihood is then computed as the negative of the complete data log-likelihood.
V.C.2 E-Step

From (V.15), it follows that the E-step of (V.11) requires conditional expectations of two forms

$$
\mathbb{E}_{X,Z|Y}[f(x^{(j)})] = \mathbb{E}_{X|Y}[f(x^{(j)})], \quad (V.16)
$$

$$
\mathbb{E}_{X,Z|Y}[z_i^{(j)} f(x^{(j)})] = \mathbb{E}_{Z|Y}[z_i^{(j)}] \mathbb{E}_{X|Y,z_i=j}[f(x^{(j)})], \quad (V.17)
$$

for some function $f$ of $x^{(j)}$, and where $\mathbb{E}_{X|Y,z_i=j}$ is the conditional expectation of $X$ given the observation $Y$ and that the $i$-th pixel belongs to layer $j$. In particular, the E-step requires

$$
\hat{x}_t^{(j)} = \mathbb{E}_{X|Y}[x_t^{(j)}], \quad \hat{P}_{t,t}^{(j)} = \mathbb{E}_{X|Y}[P_{t,t}^{(j)}], \quad \hat{P}_{t,t-1}^{(j)} = \mathbb{E}_{X|Y}[P_{t,t-1}^{(j)}], \quad (V.18)
$$

$$
\hat{x}_{t|ii}^{(j)} = \mathbb{E}_{X|Y,z_i=j}[x_t^{(j)}], \quad \hat{P}_{t|ii}^{(j)} = \mathbb{E}_{X|Y,z_i=j}[P_{t|ii}^{(j)}], \quad \hat{z}_i^{(j)} = \mathbb{E}_{Z|Y}[z_i^{(j)}].
$$

Defining, for convenience, the aggregate statistics

$$
\Gamma_i^{(j)} = \sum_{t=1}^{\tau} y_{i,t} \hat{x}_t^{(j)}, \quad \Phi_i^{(j)} = \sum_{t=1}^{\tau} \hat{P}_{t,t}^{(j)}, \quad \Phi_i^{(j)} = \sum_{t=1}^{\tau} \hat{P}_{t,t}^{(j)}, \quad \Phi_i^{(j)} = \sum_{t=1}^{\tau} \hat{P}_{t,t}^{(j)}, \quad (V.19)
$$

and substituting (V.19) and (V.15) into (V.11), leads to the $Q$ function

$$
Q(\Theta; \tilde{\Theta}) = -\frac{1}{2} \sum_{j=1}^{K} \frac{1}{r^{(j)}} \sum_{i=1}^{m} \hat{z}_i^{(j)} \left( \sum_{t=1}^{\tau} y_{i,t}^2 - 2C_i^{(j)} \Gamma_i^{(j)} + C_i^{(j)} \Phi_i^{(j)} C_i^{(j)T} \right) \quad (V.20)
$$

$$
- \frac{1}{2} \sum_{j=1}^{K} \text{tr} \left( Q^{(j)-1} \left( \hat{P}_{1,1}^{(j)} - \hat{x}_1^{(j)} \mu^{(j)} T - \mu^{(j)} (\hat{x}_1^{(j)}) T + \mu^{(j)} \mu^{(j)T} + \phi_2^{(j)} - \psi^{(j)} A^{(j)T} - A^{(j)} \psi^{(j)} T + A^{(j)} \phi_1^{(j)} A^{(j)T} \right) \right)
$$

$$
- \frac{\tau}{2} \sum_{j=1}^{K} \hat{N}_j \log r^{(j)} - \frac{\tau}{2} \sum_{j=1}^{K} \log |Q^{(j)}|.
$$

Since it is not known to which layer each pixel $y_t$ is assigned, the evaluation of the expectations of (V.18) requires marginalization over all configurations of $Z$. Hence, the $Q$ function is intractable. Two possible approximations are discussed in Sections V.D and V.E.
V.C.3 M-Step

The M-step of (V.12) updates the parameter estimates by maximizing the $Q$ function. As usual, a (local) maximum is found by taking the partial derivative with respect to each parameter and setting it to zero (see Appendix V.K.1 for complete derivation), yielding the estimates

$$C_i^{(j)*} = \Gamma_i^{(j)} \Phi_i^{(j)-1}, \quad A^{(j)*} = \psi^{(j)} \phi_i^{(j)-1}, \quad \mu^{(j)*} = \hat{x}_i^{(j)},$$

$$r^{(j)*} = \frac{1}{\tau \hat{N}_j} \sum_{i=1}^m z_i^{(j)} \left( \sum_{t=1}^\tau y_{i,t}^2 - C_i^{(j)*} \Gamma_i^{(j)} \right),$$

$$Q^{(j)*} = \frac{1}{\tau} \left( \hat{P}^{(j)}_{1,1} - \mu^{(j)*} (\mu^{(j)*})^T + \phi_2^{(j)} - A^{(j)*} \psi^{(j)} T \right).$$

The M-step of LDT learning is similar to the M-step for DT with two significant differences: 1) each row $C_i^{(j)}$ of $C^{(j)}$ is estimated separately, conditioning all statistics on the assignment of pixel $i$ to layer $j$ ($z_i = j$); 2) the estimate of the observation noise $r^{(j)}$ of each layer is a soft average of the unexplained variance of each pixel, weighted by the posterior probability $\hat{z}_i^{(j)}$ that pixel $i$ belongs to layer $j$. The EM algorithm for LDT is summarized in Algorithm 5

V.C.4 Initialization strategies

As is typical for the EM algorithm, the quality of the (locally) optimal solution depends on the initialization strategy. If an initial segmentation is available, model parameters can be initialized by learning a DT for each region, using the least-squares method. Otherwise, as in Chapter IV, we adopt a variation of the centroid splitting method of [90]. The EM algorithm is run with an increasing number of components. We start by learning an LDT with $K = 1$. A new layer is then added by duplicating the existing layer with the largest state-space noise (i.e. with the largest eigenvalue of $Q^{(j)}$). The new layer is perturbed by scaling the transition matrix $A$ by 0.99, and the resulting LDT used to initialize EM. This produces an LDT with $K = 2$. The process is repeated with the successive introduction of new layers, by perturbation of existing ones, until the desired $K$ is reached. Note that perturbing the transition matrix coerces EM to learn layers
**Algorithm 5** EM algorithm for layered dynamic textures

**Input:** video sequences $y_{1:T}$, number of layers $K$, MRF parameters $\{a_i^{(j)}, \gamma_1, \gamma_2\}$.

Initialize $\{\Theta_j\}$ for $j = 1$ to $K$.

repeat

{Expectation Step}

for $j = \{1, \ldots, K\}$ do

Compute the expectations (V.18).

Compute the aggregated statistics (V.19).

end for

{Maximization Step}

for $j = 1$ to $K$ do

Compute new layer parameters $\{\Theta_j\}$ with (V.21).

end for

until convergence

**Output:** $\{\Theta_j\}_{j=1}^K$.

with distinct dynamics.

In most cases, the approximate E-step also requires an initial estimate of the expected layer assignments $\hat{z}_i^{(j)}$. If an initial segmentation is available, it can be used as the initial $\hat{z}_i^{(j)}$. Otherwise, the initial $\hat{z}_i^{(j)}$ are estimated by approximating each pixel of the LDT with a mixture of dynamic textures. The parameters of the $j$-th mixture component are identical to the parameters of the $j$-th layer, $\{A^{(j)}, Q^{(j)}, \mu^{(j)}, C^{(j)}, r^{(j)}\}$. $\hat{z}_i^{(j)}$ is then estimated by the posterior probability that the pixel $y_i$ belongs to the $j$-th mixture component, i.e. $\hat{z}_i^{(j)} \approx p(z_i = j | y_i)$. In successive E-steps, the estimates $\hat{z}_i^{(j)}$ from the previous E-step are used to initialize the current E-step.
V.D  Approximate inference by Gibbs sampling

The expectations of (V.18) require intractable conditional probabilities. For example, \( P(X|Y) = \sum_Z P(X, Z|Y) \) requires the enumeration of all configurations of \( Z \), an operation of exponential complexity on the MRF dimensions, and intractable for even moderate frame sizes. One commonly used solution to this problem is to rely on a Gibbs sampler [91] to draw samples from the posterior distribution \( p(X, Z|Y) \), and to approximate the desired expectations by sample averages. Since, for the LDT, it is much easier to sample conditionally from the collection of variables, \( X \) or \( Z \), than from any individual \( x^{(j)} \) or \( z_i^{(j)} \), the proposed Gibbs sampler alternates between: 1) sampling \( X \sim p(X|Y, Z) \); and 2) sampling \( Z \sim p(Z|X, Y) \).

The convergence of the EM algorithm is usually monitored by tracking the likelihood \( P(Y) \) of the observed data. While this likelihood is intractable, a lower bound can be computed by summing over the configurations of \( Z \) visited by the Gibbs sampler,

\[
p(Y) = \sum_Z p(Y|Z)p(Z) \geq \sum_{Z \in Z_G} p(Y|Z)p(Z), \quad \text{(V.22)}
\]

where \( Z_G \) is the set of unique states of \( Z \) visited by the sampler. Because these tend to be the configurations of largest likelihood, the bound is a good approximation for convergence monitoring. Finally, segmentation requires the MAP solution, \( \{X^*, Z^*\} = \arg\max_{X,Z} p(X, Z|Y) \). This is computed with deterministic annealing, as in [92]. In the remainder of the section we discuss how to sample from \( p(Z|X, Y) \) and \( p(X|Y, Z) \).
V.D.1 Sampling from $p(Z|X,Y)$

The conditional distribution $p(Z|X,Y)$ can be rewritten as

$$p(Z|X,Y) = \frac{p(X,Y,Z)}{p(Y|X,Z)p(X)p(Z)} = p(Y|X,Z)p(X)p(Z) \quad (V.23)$$

$$\propto p(Y|X,Z)p(Z) \quad (V.24)$$

$$\propto \prod_{i=1}^{m} \prod_{j=1}^{K} p(y_i|x^{(j)}, z_i = j)^{z_{i}^{(j)}} \left[ \prod_{i=1}^{m} V_i(z_i) \prod_{(i,i')} V_{i,i'}(z_i, z_i') \right] \quad (V.25)$$

$$= \prod_{i=1}^{m} \prod_{j=1}^{K} \left[ \alpha_i^{(j)} p(y_i|x^{(j)}, z_i = j) \right]^{z_{i}^{(j)}} V_{i,i'}(z_i, z_i'). \quad (V.26)$$

This is identical to the MRF joint of (V.8), with modified self-potentials $\hat{\alpha}_i^{(j)} = \alpha_i^{(j)} p(y_i|x^{(j)}, z_i = j)$. Samples from this distribution can be drawn by Markov-chain Monte Carlo (MCMC) [92].

V.D.2 Sampling from $p(X|Z,Y)$

Given layer assignments $Z$, pixels are deterministically assigned to state processes. Let $I_j = \{i|z_i = j\}$ be the index set for the pixels assigned to layer $j$, and $Y_j = \{y_i|i \in I_j\}$ the corresponding set of pixel values. The joint distribution of $X$ and $Y$ factors as

$$p(X,Y|Z) = \prod_{j=1}^{K} p(x^{(j)}|Y_j,Z). \quad (V.27)$$

Each layer pair $\{x^{(j)}, Y_j\}$ is an LDS with parameters $\hat{\Theta}_j = \{A^{(j)}, Q^{(j)}, C^{(j)}, r^{(j)}, \mu^{(j)}\}$, where $\hat{C}^{(j)} = [C_i^{(j)}]_{i \in I_j}$ is the subset of the rows of the observation matrix corresponding to the pixels $Y_j$. Similarly, when conditioned on both $Y$ and $Z$, the joint distribution of $X$ factors as

$$p(X|Y, Z) = \prod_{j=1}^{K} p(x^{(j)}|Y_j, Z), \quad (V.28)$$

where $p(x^{(j)}|Y_j, Z)$ is the posterior distribution of the state sequence, under the LDS of parameters $\hat{\Theta}_j$, conditioned on the observed pixel values $Y_j$. This reduces
sampling from $p(X|Y,Z)$ to sampling a state-sequence $x^{(j)}$ from LDS $\hat{\Theta}_j$, conditioned on $Y_j$. An algorithm for efficiently drawing these sequences is given in Appendix V.K.1.

**V.E Inference by variational approximation**

Gibbs sampling is an effective method for approximate inference, but frequently too computationally intensive. A popular low-complexity alternative is to rely on a variational approximation. This consists of approximating the posterior distribution $p(X,Z|Y)$ by an approximation $q(X,Z)$ within some class of tractable probability distributions $\mathcal{F}$. Given an observation $Y$, the optimal variational approximation minimizes the Kullback-Leibler (KL) divergence between approximate and exact posteriors [93]

$$q^*(X,Z) = \text{argmin}_{q \in \mathcal{F}} D(q(X,Z) \parallel p(X,Z|Y)). \quad (V.29)$$

Note that, because the data log-likelihood $p(Y)$ is constant for an observed $Y$,

$$D(q(X,Z) \parallel p(X,Z|Y)) = \int q(X,Z) \log \frac{q(X,Z)}{p(X,Z|Y)} dXdZ \quad (V.30)$$

$$= \int q(X,Z) \log \frac{q(X,Z)p(Y)}{p(X,Y,Z)} dXdZ = \mathcal{L}(q(X,Z)) + \log p(Y), \quad (V.31)$$

where

$$\mathcal{L}(q(X,Z)) = \int q(X,Z) \log \frac{q(X,Z)}{p(X,Y,Z)} dXdZ. \quad (V.32)$$

The optimization problem of (V.29) is thus identical to

$$q^*(X,Z) = \text{argmin}_{q \in \mathcal{F}} \mathcal{L}(q(X,Z)). \quad (V.33)$$

In the remainder of the section we derive an optimal approximate factorial posterior distribution and discuss its use for inference.
V.E.1 Approximate factorial posterior distribution

The intractability of the exact posterior distribution stems from the need to marginalize over $Z$. This suggests that a tractable approximate posterior can be obtained by assuming statistical independence between pixel assignments $z_i$ and state variables $x^{(j)}$, i.e.

$$q(X, Z) = \prod_{j=1}^{K} q(x^{(j)}) \prod_{i=1}^{m} q(z_i).$$  \hspace{1cm} (V.34)

Substituting into (V.32) leads to

$$\mathcal{L}(q(X, Z)) = \int \prod_{j=1}^{K} q(x^{(j)}) \prod_{i=1}^{m} q(z_i) \log \frac{\prod_{j=1}^{K} q(x^{(j)}) \prod_{i=1}^{m} q(z_i)}{p(X, Y, Z)} dXdZ. \hspace{1cm} (V.35)$$

The $\mathcal{L}$ function (V.35) is minimized by sequentially optimizing each of the factors $q(x^{(j)})$ and $q(z_i)$, while holding the others constant [93]. This yields the factorial distributions (see Appendix V.K.1 for derivations),

$$\log q(x^{(j)}) = \sum_{i=1}^{m} h^{(j)}_i \log p(y_i | x^{(j)}, z_i = j) + \log p(x^{(j)}) - \log Z_q^{(j)}, \hspace{1cm} (V.36)$$

$$\log q(z_i) = \sum_{j=1}^{K} z^{(j)}_i \log h^{(j)}_i, \hspace{1cm} (V.37)$$
where \( Z_q^{(j)} \) is a normalization constant (Appendix V.K.1), \( h_i^{(j)} \) are the variational parameters

\[
h_i^{(j)} = E[z_i^{(j)}] = \frac{\alpha_i^{(j)} g_i^{(j)}}{\sum_{k=1}^K \alpha_i^{(k)} g_i^{(k)}}, \tag{V.38}
\]

\[
\log g_i^{(j)} = E_{z^{(j)}} \left[ \log p(y_i | x^{(j)}, z_i = j) \right] + \sum_{(i,i') \in E} h_{i'}^{(j)} \log \frac{\gamma_1}{\gamma_2}, \tag{V.39}
\]

and \( E_{z^{(j)}} \) and \( E_{z_i} \) expectations with respect to \( q(x^{(j)}) \) and \( q(z_i) \).

The optimal factorial distributions can be interpreted as follows. The variational parameters \( \{h_i^{(j)}\} \), which appear in both \( q(z_i) \) and \( q(x^{(j)}) \), account for the dependence between \( X \) and \( Z \) (see Figure V.3). \( h_i^{(j)} \) is the posterior probability of assigning pixel \( y_i \) to layer \( j \), and is estimated by the expected log-likelihood of observing pixel \( y_i \) from layer \( j \), with an additional boost of \( \log \frac{\gamma_1}{\gamma_2} \) per neighboring pixel also assigned to layer \( j \). \( h_i^{(j)} \) also weighs the contribution of each pixel \( y_i \) to the factor \( q(x^{(j)}) \), which effectively acts as a soft assignment of pixel \( y_i \) to layer \( j \). Also note that in (V.36), \( h_i^{(j)} \) can be absorbed into \( p(y_i | x^{(j)}, z_i = j) \), making \( q(x^{(j)}) \) the distribution of an LDS parameterized by \( \tilde{\Theta}_j = \{A^{(j)}, Q^{(j)}, C^{(j)}, R_j, \mu^{(j)}\} \), where \( R_j \) is a diagonal matrix with entries \( \frac{\gamma_1}{h_i^{(j)}}, \ldots, \frac{\gamma_1}{h_i^{(j)}} \). Finally \( \log g_i^{(j)} \) is computed by rewriting (V.39) as

\[
\log g_i^{(j)} = E_{x^{(j)}} \left[ -\frac{1}{2r_{(j)}} \sum_{t=1}^\tau \| y_{i,t} - C_i^{(j)} x_t^{(j)} \|^2 - \frac{\tau}{2} \log 2\pi r_{(j)} \right] + \sum_{(i,i') \in E} h_{i'}^{(j)} \log \frac{\gamma_1}{\gamma_2}
\]

\[
= -\frac{1}{2r_{(j)}} \left( \sum_{t=1}^\tau y_{i,t}^2 - 2C_i^{(j)} \sum_{t=1}^\tau y_{i,t} E_{x^{(j)}}[x_t^{(j)}] + C_i^{(j)} \sum_{t=1}^\tau E_{x^{(j)}}[x_t^{(j)} x_t^{(j)T}] C_i^{(j)T} \right) - \frac{\tau}{2} \log 2\pi r_{(j)} + \sum_{(i,i') \in E} h_{i'}^{(j)} \log \frac{\gamma_1}{\gamma_2}, \tag{V.41}
\]

where the expectations \( E_{x^{(j)}}[x_t^{(j)}] \) and \( E_{x^{(j)}}[x_t^{(j)} x_t^{(j)T}] \) are computed with the Kalman smoothing filter (Section II.C) for an LDS with parameters \( \tilde{\Theta}_j \).

The optimal \( q^*(X, Z) \) is found by iterating through each pixel \( i \), recomputing the variational parameters \( h_i^{(j)} \) according to (V.38) and (V.39), until con-
Algorithm 6 Variational Approximation for LDT

1. **Input**: LDT parameters $\Theta$, batches $\{B_1, \ldots, B_M\}$.

2. Initialize $\{h_i^{(j)}\}$.

3. **repeat**
   4. {Recompute variational parameters for each batch}
   5. **for** $B \in \{B_1, \ldots, B_M\}$ **do**
   6. compute $E_{x(j)}[x_t^{(j)}]$ and $E_{x(j)}[x_t^{(j)}x_t^{(j)T}]$ by running the Kalman smoothing filter with parameters $\tilde{\Theta}_j$, for $j = \{1, \ldots, K\}$.
   7. **for** $i \in B$ **do**
   8. compute $\log g_i^{(j)}$ using (V.41), for $j = \{1, \ldots, K\}$.
   9. compute $h_i^{(j)}$ using (V.38), for $j = \{1, \ldots, K\}$.
   10. **end for**
   11. **end for**
   12. **until** convergence of $g_i^{(j)}$ and $h_i^{(j)}$
   13. **return** the approximate posterior $q^*(X, Z)$.

V.E.2 Approximate inference

In the remainder of the section we discuss inference with the approximate posterior $q^*(X, Z)$. 

vergence. This might be computationally expensive, because it requires running a Kalman smoothing filter for each pixel. The computational load can be reduced by updating batches of variational parameters at a time. In this work, we define a batch $B$ as the set of nodes in the MRF with non-overlapping Markov blankets (as in [94]), i.e. $B = \{i | (i, i') \notin \mathcal{E}, \forall i' \in B\}$. In practice, batch updating typically converges to the solution reached by serial updating, but is significantly faster.

The variational approximation using batch (synchronous) updating is summarized in Algorithm 6.
E-step

In (V.18), expectations with respect to $p(X|Y)$ and $p(Z|Y)$ can be estimated as

$$
\hat{x}_t^{(j)} \approx \mathbb{E}_{x^{(j)}}[x_t^{(j)}], \quad \hat{P}_{t,t}^{(j)} \approx \mathbb{E}_{x^{(j)}}[x_t^{(j)} x_t^{(j)T}],
$$

where $\mathbb{E}_{x^{(j)}}$ is the expectation with respect to $q^*(x^{(j)})$. The remaining expectations of (V.18) are with respect to $p(X|Y, z_i = j)$, and can be approximated with $q^*(X|z_i = j)$ by running the variational algorithm with a binary $h_i^{(j)}$, set to enforce $z_i = j$. Note that if $m$ is large (as is the case with video), fixing the value of a single $z_i = j$ will have little effect on the posterior, due to the combined evidence from the large number of other pixels in the layer. Hence, expectations with respect to $p(X|Y, z_i = j)$ can also be approximated with $q^*(X)$ when $m$ is large, i.e.

$$
\hat{x}_t^{(j)} \approx \mathbb{E}_{x^{(j)}|z_i = j}[x_t^{(j)}] \approx \mathbb{E}_{x^{(j)}}[x_t^{(j)}], \quad \hat{P}_{t,t|t}^{(j)} \approx \mathbb{E}_{x^{(j)}|z_i = j}[x_t^{(j)} x_t^{(j)T}] \approx \mathbb{E}_{x^{(j)}}[x_t^{(j)} x_t^{(j)T}],
$$

where $\mathbb{E}_{x^{(j)}|z_i = j}$ is the expectation with respect to $q^*(x^{(j)}|z_i = j)$. Finally, we note that the EM algorithm with variational E-step is guaranteed to converge. However, the approximate E-step prevents convergence to local maxima of the data log-likelihood [95]. Despite this limitation, the algorithm still performs well empirically, as shown in Section V.G.

Lower bound on $p(Y)$

Convergence is monitored with a lower-bound on $p(Y)$, which follows from the non-negativity of the KL divergence and (V.31),

$$
D(q(X, Z) \| p(X, Z|Y)) = \mathcal{L}(q(X, Z)) + \log p(Y) \geq 0 \quad \text{(V.44)}
$$

$$
\Rightarrow \log p(Y) \geq -\mathcal{L}(q(X, Z)). \quad \text{(V.45)}
$$
Evaluating $L$ for the optimal $q^*$ (see Appendix V.K.1 for derivation), the lower bound is

$$
\log p(Y) \geq \sum_j \log Z_q^{(j)} - \sum_{j,i} h_i^{(j)} \log \frac{h_i^{(j)}}{\alpha_i^{(j)}}
$$

(V.46)

$$
+ \sum_{(i,i') \in E} \left( \log \gamma_2 + \sum_{j} h_i^{(j)} h_i^{(j')} \log \frac{\gamma_1}{\gamma_2} \right) - \log Z_Z.
$$

MAP layer assignment

Given the observed video $Y$, the maximum a posteriori layer assignment $Z$ (i.e. segmentation) is

$$
Z^* = \arg\max_Z p(Z|Y) = \arg\max_Z \int p(X, Z|Y) dX 
$$

(V.47)

$$
\approx \arg\max_Z \int q^*(X, Z) dX = \arg\max_Z \int \prod_{j=1}^K q^*(x^{(j)}) \prod_{i=1}^m q^*(z_i) dX 
$$

(V.48)

$$
= \arg\max_Z \prod_{i=1}^m q^*(z_i).
$$

(V.49)

Hence, the MAP solution for $Z$ is approximated by the individual MAP solutions for $z_i$, i.e.

$$
z_i^* \approx \arg\max_j h_i^{(j)}, \quad \forall i.
$$

(V.50)

V.F Temporally-switching layered dynamic textures

In this section, we explore an extension of the LDT which allows switching of the MRF in time. We denote the new model as a temporally-switching layered dynamic texture (TS-LDT). As with the original LDT, each of the $K$ layers has a state process $x^{(j)} = \{x_t^{(j)}\}$ that evolves separately. A pixel trajectory $y_i = \{y_{i,t}\}$ is assigned to one of the layers, at each time instance, through the hidden variable $z_{i,t}$. The collection of hidden variables $Z = \{z_{i,t}\}$ is modeled as a Markov random field (MRF) to ensure spatial and temporal smoothness of the layer assignments.
The model equations are
\[
\begin{align*}
  x_t^{(j)} &= A^{(j)} x_{t-1}^{(j)} + v_t^{(j)} , \quad j \in \{1, \cdots, K\} \\
  y_{i,t} &= C_i^{(z_{i,t})} x_t^{(z_{i,t})} + w_{i,t} + \gamma_i^{(z_{i,t})} , \quad i \in \{1, \cdots, N\}
\end{align*}
\] (V.51)
where $C_i^{(j)} \in \mathbb{R}^{1 \times n}$, $v_t^{(j)} \sim \mathcal{N}(0, Q^{(j)})$, and $x_t^{(j)} \sim \mathcal{N}(\mu^{(j)}, Q^{(j)})$ as before with the LDT. For the TS-LDT, the observation noise processes is now distributed as $w_{i,t} \sim \mathcal{N}(0, r^{(z_{i,t})})$, and we have also included the mean value, $\gamma_i^{(j)} \in \mathbb{R}$, for pixel $i$ in layer $j$. Note that we must specify the mean for each layer, since a given pixel may switch between layers at any given time.

**V.F.1 Joint distribution of the TS-LDT**

Similar to Section V.B.1, we introduce an indicator variable $z_{i,t}^{(j)}$ of value 1 if and only if $z_{i,t} = j$, and 0 otherwise. Under the assumption that the state processes $X$ and layer assignments $Z$ are independent, the joint distribution factors as

\[
p(X, Y, Z) = p(Y | X, Z)p(X)p(Z) = \prod_{i=1}^{m} \prod_{j=1}^{K} \prod_{t=1}^{\tau} p(y_{i,t} | x_t^{(j)}, z_{i,t} = j)^{z_{i,t}^{(j)}} \prod_{j=1}^{K} p(x^{(j)})p(Z),
\] (V.53)

where the conditional observation likelihood is

\[
p(y_{i,t} | x_t^{(j)}, z_{i,t} = j) = G(y_{i,t}, C_i^{(j)} x_t^{(j)} + \gamma_i^{(j)}, r^{(j)}),
\] (V.54)

and the distribution for $p(x^{(j)})$ is the same as the LDT, given in (V.4, V.5). Finally, for the layer assignments $Z$, we assume that each frame $Z_t = \{z_{i,t}\}_{i=1}^{m}$ has the same MRF structure (e.g. Figure V.2), with temporal edges only connecting nodes corresponding to the same pixel (e.g. $z_{i,t}$ and $z_{i,t+1}$). The layer assignments are then jointly distributed as

\[
p(Z) = \frac{1}{Z} \left[ \prod_{t=1}^{\tau} \prod_{i=1}^{m} V_{i,t}(z_{i,t}) \right] \left[ \prod_{t=1}^{\tau} \prod_{(i,i') \in E_t} V_{i,i'}(z_{i,t}, z_{i',t}) \right] \left[ \prod_{i=1}^{m} \prod_{(t,t') \in E_i} V_{t,t'}(z_{i,t}, z_{i,t'}) \right],
\]
where \( \mathcal{E}_t \) is the set of MRF in frame \( t \), \( \mathcal{E}_i \) is the set of MRF edges between frames for pixel \( i \), and \( \mathcal{Z} \) a normalization constant (partition function). The potential functions \( V_{i,t}, V_{i,i'}, V_{t,t'} \) are of the form:

\[
V_{i,t}(z_{i,t}) = \prod_{j=1}^{K}(\alpha_{i,t}^{(j)})^{z_{i,t}^{(j)}} = \begin{cases} 
\alpha_{i,t}^{(1)}, & z_{i,t} = 1 \\
\vdots & \\
\alpha_{i,t}^{(K)}, & z_{i,t} = K 
\end{cases}, \quad (V.55)
\]

\[
V_{i,i'}(z_{i,t}, z_{i',t}) = \gamma_2 \prod_{j=1}^{K} \left( \frac{\gamma_1}{\gamma_2} \right)^{z_{i,t}^{(j)}z_{i',t}^{(j)}} = \begin{cases} 
\gamma_1, & z_{i,t} = z_{i',t} \\
\gamma_2, & z_{i,t} \neq z_{i',t} 
\end{cases}, \quad (V.56)
\]

\[
V_{t,t'}(z_{i,t}, z_{i',t'}) = \beta_2 \prod_{j=1}^{K} \left( \frac{\beta_1}{\beta_2} \right)^{z_{i,t}^{(j)}z_{i',t'}^{(j)}} = \begin{cases} 
\beta_1, & z_{i,t} = z_{i,t'} \\
\beta_2, & z_{i,t} \neq z_{i,t'} 
\end{cases}, \quad (V.56)
\]

where \( V_{i,t} \) is the prior probability of each layer in each frame \( t \), and \( V_{i,i'} \) and \( V_{t,t'} \) attributes higher probability to configurations with neighboring pixels (both spatially and temporally) in the same layer.

**V.F.2 Parameter estimation with the EM algorithm**

The EM algorithm for the TS-LDT is similar to that of the LDT (derivation appears in Appendix V.K.2). For the E-step, we compute the expectations, now conditioned on \( z_{i,t} = j \),

\[
\hat{x}_t^{(j)} = \mathbb{E}_{X|Y}[x_t^{(j)}], \quad \hat{p}_t^{(j)} = \mathbb{E}_{X|Y}[P_{t,t}^{(j)}], \quad \hat{p}_{t,t-1}^{(j)} = \mathbb{E}_{X|Y}[P_{t,t-1}^{(j)}], \\
\hat{x}_{t|i}^{(j)} = \mathbb{E}_{X|z_i = j}[x_t^{(j)}], \quad \hat{p}_{t|i}^{(j)} = \mathbb{E}_{X|y,z_i = j}[P_{t,t}^{(j)}], \quad \hat{z}_{i,t}^{(j)} = \mathbb{E}_{Z|Y}[z_{i,t}^{(j)}],
\]

where \( \mathbb{E}_{X|Y,z_i = j} \) is the conditional expectation of \( X \) given the observation \( Y \) and that the \( i \)-th pixel at time \( t \) belongs to layer \( j \). Next, the aggregated statistics are computed

\[
\phi_1^{(j)} = \sum_{t=1}^{T-1} \hat{p}_t^{(j)}, \quad \phi_2^{(j)} = \sum_{t=2}^{T} \hat{p}_t^{(j)}, \quad \psi^{(j)} = \sum_{t=2}^{T} \hat{p}_{t,t-1}^{(j)}, \\
\Phi_1^{(j)} = \sum_{t=1}^{T} \hat{z}_{i,t}^{(j)} \hat{p}_t^{(j)}, \quad \Gamma_i^{(j)} = \sum_{t=1}^{T} \hat{z}_{i,t}^{(j)} (y_{i,t} - \gamma_i^{(j)}) \hat{x}_{t|i}^{(j)}, \\
\check{N}_j = \sum_{t=1}^{T} \sum_{i=1}^{m} \hat{z}_{i,t}^{(j)} \hat{x}_{t|i}^{(j)} \check{N}_i = \sum_{t=1}^{T} \sum_{i=1}^{m} \hat{z}_{i,t}^{(j)} \hat{x}_{t|i}^{(j)}.
\]

(V.58)
In the M-step, the TS-LDT parameters are updated according to

\[ C_i^{(j)*} = \Gamma_i^{(j)*} \Phi_i^{(j)*-1} , \quad r_i^{(j)*} = \frac{1}{N_i} \sum_{i=1}^{m} \left[ \sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)} (y_{i,t} - \gamma_i^{(j)})^2 - C_i^{(j)*} \Gamma_i^{(j)*} \right] , \]

\[ A_j^{(j)*} = \psi_j^{(j)} \phi_j^{(j)*-1} , \quad Q_j^{(j)*} = \frac{1}{\tau} \left[ \hat{P}_j^{(j)} - \mu_j^{(j)} (\mu_j^{(j)*})^T + \phi_j^{(j)} - A_j^{(j)*} \psi_j^{(j)*} \right] , \quad (V.59) \]

\[ \mu_i^{(j)*} = \hat{x}_i^{(j)} , \quad \gamma_i^{(j)*} = \frac{1}{\sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)}} \left( \sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)} y_{i,t} - C_i^{(j)} \xi_i^{(j)} \right) , \]

which now take into account the mean of each layer \( \gamma_i^{(j)} \).

**V.F.3  Inference by variational approximation**

Similar to the LDT, the variational approximation for the TS-LDT assumes statistical independence between pixel assignments \( z_{i,t} \) and state variables \( x^{(j)} \), i.e.

\[ q(X, Z) = \prod_{j=1}^{K} q(x^{(j)}) \prod_{i=1}^{m} \prod_{t=1}^{\tau} q(z_{i,t}), \quad (V.60) \]

resulting in factorial distributions (see Appendix V.K.2 for derivation)

\[ \log q(x^{(j)}) = \sum_{t=1}^{\tau} \sum_{i=1}^{m} h_{i,t}^{(j)} \log p(y_{i,t}|x_{i,t}^{(j)}, z_{i,t} = j) + \log p(x^{(j)}) \quad (V.61) \]

\[ \log q(z_{i,t}) = \sum_{j=1}^{K} z_{i,t}^{(j)} \log h_{i,t}^{(j)} \quad (V.62) \]

where \( Z_q^{(j)} \) is a normalization constant (Appendix V.K.2), \( h_{i,t}^{(j)} \) are the variational parameters

\[ h_{i,t}^{(j)} = E_{z_{i,t}}[z_{i,t}^{(j)}] = \frac{\alpha_{i,t}^{(j)} g_{i,t}^{(j)}}{\sum_{k=1}^{K} \alpha_{i,t}^{(k)} g_{i,t}^{(k)}}, \quad (V.63) \]

\[ \log g_{i,t}^{(j)} = E_{x_{i,t}^{(j)}} \left[ \log p(y_{i,t}|x_{i,t}^{(j)}, z_{i,t} = j) \right] \]

\[ + \sum_{(i',j') \in E_i} h_{i',j'}^{(j)} \log \frac{\gamma_1}{\gamma_2} + \sum_{(i',j') \in E_i} h_{i',j'}^{(j)} \log \frac{\beta_1}{\beta_2}, \quad (V.64) \]
and $E_{x_t^{(j)}}$ and $E_{z_t,t}$ expectations with respect to $q(x_t^{(j)})$ and $q(z_{i,t})$. Finally $log g_{i,t}^{(j)}$ is computed by rewriting (V.64) as

$$
\log g_{i,t}^{(j)} = E_{x_t^{(j)}} \left[ \frac{-1}{2r(j)} \left\| y_{i,t} - \gamma_i^{(j)} - C_i^{(j)} x_t^{(j)} \right\|^2 - \frac{1}{2} \log 2\pi r(j) \right] \quad \text{(V.65)}
$$

$$+ \sum_{(i',i') \in \mathcal{E}_t} h_{i',t}^{(j)} \log \gamma_1^{-1} + \sum_{(t',t') \in \mathcal{E}_t} h_{i',t}^{(j)} \log \beta_1 \beta_2^{-1}
$$

$$= \frac{-1}{2r(j)} \left( (y_{i,t} - \gamma_i^{(j)})^2 - 2C_i^{(j)} (y_{i,t} - \gamma_i^{(j)}) E_{x_t^{(j)}} [x_t^{(j)}] \right. \left. + C_i^{(j)} E_{x_t^{(j)}} [x_t^{(j)} x_t^{(j)\top}] C_i^{(j)T} \right) - \frac{1}{2} \log 2\pi r(j)
$$

$$+ \sum_{(i',i') \in \mathcal{E}_t} h_{i',t}^{(j)} \log \gamma_1 \gamma_2^{-1} + \sum_{(t',t') \in \mathcal{E}_t} h_{i',t}^{(j)} \log \beta_1 \beta_2^{-1},
$$

where the expectations $E_{x_t^{(j)}} [x_t^{(j)}]$ and $E_{x_t^{(j)}} [x_t^{(j)} x_t^{(j)\top}]$ are computed with the Kalman smoothing filter for an LDS with parameters $\tilde{\Theta}_j = \{A^{(j)}, Q^{(j)}, C^{(j)}, R_t^{(j)}, \mu^{(j)}\}$, with an observation noise $R_t^{(j)}$ that is diagonal with entries $[r_{1,1}^{(j)}, \cdots, r_{h_{1,1}^{(j)},h_{1,1}^{(j)}}^{(j)}]$ that changes at each time step.

The variational approximation is computed by sequentially optimizing each of the factors $q(x^{(j)})$ and $q(z_{i,t})$, while holding the remaining constant, by recomputing the appropriate variational parameters. The optimal approximate posterior $q^*(X, Z)$ can be used for the following inference problems:

- **E-Step**: The expectations with respect to $p(X|Y)$ and $p(Z|Y)$ in (V.57), can be approximated as

$$
\hat{x}_t^{(j)} \approx E_{x_t^{(j)}} [x_t^{(j)}], \quad \hat{P}_{t,t}^{(j)} \approx E_{x_t^{(j)}} [x_t^{(j)} x_t^{(j)\top}],
$$

$$
\hat{z}_{i,t}^{(j)} \approx h_{i,t}^{(j)}, \quad \hat{P}_{t,t-1}^{(j)} \approx E_{x_t^{(j)}} [x_t^{(j)} x_{t-1}^{(j)\top}],
$$

$$
\hat{x}_{t|1}^{(j)} \approx E_{x_t^{(j)}|z_{i,t}=j} [x_t^{(j)}] \approx E_{x_t^{(j)}} [x_t^{(j)}],
$$

$$
\hat{P}_{t,t|1}^{(j)} \approx E_{x_t^{(j)}|z_{i,t}=j} [x_t^{(j)} x_t^{(j)\top}] \approx E_{x_t^{(j)}} [x_t^{(j)} x_t^{(j)\top}],
$$

where $E_{x_t^{(j)}}$ is the expectation with respect to $q^*(x^{(j)})$. 
• **Lower-bound on** $p(Y)$: Using (V.45),

$$
\log p(Y) \geq \sum_j \log Z_q^{(j)} - \sum_{j,i,t} h_{i,t}^{(j)} \log \frac{h_{i,t}^{(j)}}{\alpha_{i,t}^{(j)}} 
+ \sum_t \sum_{(i,i') \in E_t} \left( \log \gamma_2 + \sum_j h_{i,t}^{(j)} h_{i',t}^{(j)} \log \frac{\gamma_1}{\gamma_2} \right)
+ \sum_i \sum_{(t,t') \in E_i} \left( \log \beta_2 + \sum_j h_{i,t}^{(j)} h_{i',t}^{(j)} \log \frac{\beta_1}{\beta_2} \right) - \log Z.
$$

(V.68)

• **MAP layer assignment**: The MAP solution for $Z$ is approximated by the individual MAP solutions for $z_{i,t}$, i.e.

$$
z_{i,t}^* \approx \arg\max_j h_{i,t}^{(j)}, \ \forall i.
$$

(V.69)

V.G Experimental evaluation of approximate inference and EM

In this section, we present a quantitative comparison of approximate inference on a synthetic dataset, along with a comparison in the context of EM learning.

V.G.1 Synthetic dataset

A synthetic dataset of LDT samples was generated as follows. A number of LDTs of $K = 2$ components was produced by randomly sampling parameter values for each component $j = \{1, 2\}$, according to

$$
r^{(j)} \sim W(1, 1), \ \lambda_0^{(j)} \sim U_1(0, 1), \ A_0^{(j)} \sim N_{n,n}(0, 1), \ A^{(j)} = \lambda_0^{(j)} A_0^{(j)} / \lambda_{\max}(A_0^{(j)}),$

$$
S^{(j)} = Q^{(j)}, \ \ Q^{(j)} \sim W(I_n, n), \ \mu^{(j)} \sim U_n(-5, 5), \ C^{(j)} \sim N_{m,n}(0, 1),$
\]

where $N_{m,n}(\mu, \sigma^2)$ is a distribution on $\mathbb{R}^{m \times n}$ matrices with each entry distributed as $N(\mu, \sigma^2)$, $W(\Sigma, d)$ is a Wishart distribution with covariance $\Sigma$ and $d$ degrees of freedom, $U_d(a, b)$ is a distribution on $\mathbb{R}^d$ vectors with each coordinate distributed
uniformly between $a$ and $b$, and $\lambda_{\text{max}}(A_0^{(j)})$ is the magnitude of the largest eigenvalue of $A_0^{(j)}$. Note that $A^{(j)}$ is a random scaling of $A_0^{(j)}$ such that the system is stable (i.e. the poles of $A^{(j)}$ are within the unit circle). The MRF used a 4-connected neighborhood, with parameters $\log \gamma_1 = -\log \gamma_2 = 0.4$ and $\log \alpha_i^{(j)} = 0 \forall i, j$. A set of 200 LDT parameters was sampled for all combinations of $n = \{10, 15, 20\}$ and $m = \{600, 1200\}$ (corresponding to a grid size of $30 \times 20$ and $40 \times 30$), and a time-series sample $\{X,Y,Z\}$, with temporal length 75, was drawn from each LDT, forming a synthetic dataset of 1200 samples. Finally, additional datasets, each with 1200 samples, were formed by repeating with $K = \{3, 4\}$.

V.G.2 Inference experiments

In this experiment, we compare the variational approximation (which we denote as “Var”) with Gibbs sampling (Gibbs). For Gibbs, 100 samples were generated (25 samples from 4 trials), using a “burn-in” phase of 100 iterations, and 5 iterations between each sample draw. Each inference method was initialized with the DTM approximation discussed in Section V.C.4. The conditional means of the hidden variables, $\hat{z}_i = \mathbb{E}(z_i|Y)$ and $\hat{x}_t^{(j)} = \mathbb{E}(x_t^{(j)}|Y)$, were estimated, and the standard deviations with respect to the ground-truth values of $z_i$ and $x_t^{(j)}$ were computed. The average value of the lower-bound $\hat{L}$ of the log-likelihood $\log P(Y)$ was also computed, along with the Rand index [82] between the true segmentation $Z$ and the approximate MAP solution $\hat{Z}$. The Rand index is a measure of clustering performance, and intuitively is the probability of pair-wise agreement between the clustering and the ground-truth. Finally, the performance metrics were averaged over the synthetic dataset for $K = 2$.

Table V.1 Comparison of approximate inference methods on synthetic data.

<table>
<thead>
<tr>
<th></th>
<th>std($\hat{z}_i$)</th>
<th>std($\hat{x}_t^{(j)}$)</th>
<th>mean($\hat{L}$)</th>
<th>$\hat{Z}$ Rand</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var</td>
<td>0.062</td>
<td>0.327</td>
<td>-1.437e5</td>
<td>0.995</td>
<td>10.9s</td>
</tr>
<tr>
<td>Gibbs</td>
<td>0.062</td>
<td>0.330</td>
<td>-1.437e5</td>
<td>0.995</td>
<td>456s</td>
</tr>
<tr>
<td>DTM</td>
<td>0.158</td>
<td>3.883</td>
<td>-2.849e5</td>
<td>0.933</td>
<td>64.3s</td>
</tr>
</tbody>
</table>
The estimation errors of the two approximate inference algorithms are presented in Table V.1. Var and Gibbs have comparable performance, with the exception of a slight difference in the estimates of $x^{(j)}_t$. However, Var is significantly faster than Gibbs, with a speed up of over 40 times. Finally, although the estimation error of the DTM approximation is large for $\hat{x}^{(j)}_t$, the error of the layer assignments $\hat{z}_i$ is reasonable. This makes the DTM approximation a suitable initialization for the other inference algorithms.

V.G.3 EM experiments

We next compare approximate inference in the context of the EM algorithm. LDT models were learned from the observed $Y$, using EM with the two approximate E-steps, which we denote “VarEM” and “GibbsEM”. The LDTs learned from the two EM algorithms were compared via their segmentation performance: the MAP solution $\hat{Z}$ was compared with the ground-truth $Z$ using the Rand index. Finally, the Rand index was averaged over all LDTs in each synthetic dataset $K = \{2, 3, 4\}$.

Figure V.4 presents the plots of Rand index versus the median run-time.
obtained for each method. VarEM and GibbsEM perform comparably (Rand of 0.998) for \( K = 2 \). However, GibbsEM outperforms VarEM when \( K = \{3, 4\} \), with Rand 0.959 and 0.929 versus 0.923 and 0.881, respectively. This difference is due to the uni-modality of the approximate variational posterior; given multiple possible layer assignments (posterior modes), the variational approximation can only account for one of the configurations, effectively ignoring the other possibilities. While this behavior is acceptable when computing MAP assignments of a learned LDT (e.g. the inference experiments in Section V.G.2), it may be detrimental for LDT learning. VarEM is not allowed to explore multiple configurations, which may lead to convergence to a poor local maximum. Poor performance of VarEM is more likely when there are multiple possible configurations, i.e. when \( K \) is large (empirically, when \( K \geq 3 \)). However, the improved performance of GibbsEM comes at a steep computational cost, with run-times that are 150 to 250 times longer than those of VarEM. In practice, the performance of VarEM can be improved by initializing EM at a good starting point. For example, when applying the LDT to motion segmentation, in Section V.H, we will initialize the VarEM using the segmentation results of the mixture of dynamic textures. Finally, for comparison, the data was segmented with the generalized PCA (GPCA) method of [17], which is shown to perform worse than both VarEM and GibbsEM for all \( K \). This is most likely due to the “noiseless” assumption of the underlying model, which makes the method susceptible to outliers, or other stochastic variations.

V.H Application to motion segmentation

In this section, we present experiments on motion segmentation of both synthetic and real video using the LDT. All segmentations were obtained by learning an LDT with the EM algorithm, and computing the posterior layer assignments \( \hat{Z} = \arg\max_Z p(Z|Y) \). Due to the significant computational cost of Gibbs sampling, we only report on the variational E-step. In all cases, EM was initialized
as discussed in Section V.G. We also compare the LDT segmentations with those produced by various state-of-the-art methods in the literature, including DTM with a patch-size of $5 \times 5$, generalized PCA (GPCA) [17], and level-sets with Ising models [19] (for $K = 2$ only). Segmentations are evaluated by computing the Rand index [82] with the ground-truth. We start by presenting results on synthetic textures containing different types of circular motion, and then present a quantitative evaluation on a large synthetic texture database used in Chapter IV. Finally, we present results on real-world video sequences. Videos of the results are available in the supplemental material [52].

V.H.1 Results on synthetic circular motion

We first demonstrate LDT segmentation of sequences with different types of circular motion. These experiments were based on video containing several rings of distinct circular motion, as shown in Figure V.5a. Each video sequence $I_{x,y,t}$ has dimensions $101 \times 101$, and was generated according to

$$I_{x,y,t} = 128 \cos(c_r \theta + \frac{2\pi}{T_r}t + v_t) + 128 + w_t,$$  \hspace{1cm} (V.70)
where \( \theta = \arctan\left(\frac{y-51}{x-51}\right) \) is the angle of the pixel \((x, y)\) relative to the center of the video frame, \( v_t \sim \mathcal{N}(0, (2\pi/50)^2) \) is the phase noise, and \( w_t \sim \mathcal{N}(0, 10^2) \) is the observation noise. The parameter \( T_r \in \{5, 10, 20, 40\} \) determines the speed of each ring, while \( c_r \) determines the number of times the texture repeats around the ring. Here, we select \( c_r \) such that all the ring textures have the same spatial period. Sequences were generated with \( \{2, 3, 4\} \) circular or square rings, with a constant center patch (see Figure V.5 left and middle). Finally, a third set of dynamics was created, by allowing the textures to move only horizontally or vertically (see Figure V.5 right).

The sequences were segmented with LDT, DTM, and GPCA with \( n = 2 \), producing the results shown in Figure V.5 (b,c, and d). LDT (Figure V.5b) correctly segments all the rings, favoring global homogeneity over localized grouping of segments by texture orientation. On the other hand, DTM (Figure V.5c) tends to find incorrect segmentations based on local direction of motion. In addition, DTM sometimes incorrectly assigns one segment to the boundaries between rings, illustrating how the poor boundary accuracy of the patch-based segmentation framework can create substantial problems. Finally, GPCA (Figure V.5d) is able to correctly segment 2 rings, but fails when there are more. In these cases, GPCA correctly segments one of the rings, but randomly segments the remainder of the video. These results illustrate how LDT can correctly segment sequences whose motion is globally (at the ring level) homogeneous, but locally (at the patch level) heterogeneous. Both DTM and GPCA fail to exhibit this property. Quantitatively, this is reflected by the much higher average Rand scores of the segmentations produced by LDT (1.00, as compared to 0.482 and 0.826 for DTM and GPCA, respectively).

\(^1\text{Ising [19] could not be applied since there are more than 2 segments.}\)
Table V.2 Average Rand index for various segmentation algorithms on the synthetic texture database.

<table>
<thead>
<tr>
<th>Method</th>
<th>$K = 2$</th>
<th>$K = 3$</th>
<th>$K = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDT</td>
<td>0.944 (05)</td>
<td>0.894 (12)</td>
<td>0.916 (20)</td>
</tr>
<tr>
<td>DTM</td>
<td>0.912 (17)</td>
<td>0.844 (15)</td>
<td>0.857 (15)</td>
</tr>
<tr>
<td>Ising [19]</td>
<td>0.927 (12)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>AR [19]</td>
<td>0.922 (10)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>AR0 [19]</td>
<td>0.917 (20)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>GPCA [17]</td>
<td>0.538 (02)</td>
<td>0.518 (10)</td>
<td>0.538 (10)</td>
</tr>
</tbody>
</table>

V.H.2 Results on synthetic texture database

We next present results on the synthetic texture database used in Chapter IV, which contains 299 sequences with $K = \{2, 3, 4\}$ regions of different video textures (e.g. water, fire, vegetation), as illustrated in Figure V.7a. In Chapter IV, the database was segmented with DTM, using a fixed initial contour. Although DTM was shown to be superior to other state-of-the-art methods [19, 17], the segmentations contain some errors due to the poor boundary localization discussed above. In this experiment, we show that using the LDT to refine the segmentations produced by DTM substantially improves the results from Chapter IV. For comparison, we apply the level-set methods of [19] (Ising, AR, and AR0) using identical DTM initialization. The database was also segmented with GPCA [17], which does not require any initialization. Each method was run for several values of $n$, and the average Rand index was computed for each $K$. No post-processing was applied to the segmentations.

Table V.2 shows the performance obtained, with the best $n$, by each algorithm. It is clear that LDT segmentation significantly improves the initial segmentation produced by DTM: the average Rand increases from 0.912 to 0.944, from 0.844 to 0.894, and from 0.857 to 0.916, for $K = \{2, 3, 4\}$ respectively. LDT also performs best among all algorithms, with Ising as the closest competitor (Rand 0.927). Figure V.6a shows a plot of the Rand index versus the dimension $n$ of the segmentation models, demonstrating that LDT segmentation is robust to the
choice of \( n \).

We also examine the segmentation precision of the various methods. Given a threshold \( \theta \), segmentation precision is defined as the percentage of segmentations deemed to be correct with respect to the threshold, i.e. the percentage with Rand index larger than \( \theta \). Figure V.6b plots the precision of the segmentation algorithms for different threshold levels. Segmentation with LDT shows a dramatic improvement in the precision curve. Qualitatively, LDT improves the DTM segmentation in three ways: 1) segmentation boundaries are more precise, due to the region-level modeling (rather than patch-level); 2) segmentations are less noisy, due to the inclusion of the MRF prior; and 3) gross errors, e.g. texture borders marked as segments, are eliminated. Several examples of these improvements are presented in Figure V.7b and Figure V.7c. From left to right, the first example is a case where the LDT corrects a noisy DTM segmentation (imprecise boundaries and spurious segments). The second and third examples are cases where the DTM produces a poor segmentation (e.g. the border between two textures erroneously marked as a segment), which the LDT corrects. The final two examples are very difficult cases. In the fourth example, the initial DTM segmentation is very poor. Albeit a substantial improvement, the LDT segmentation is still noisy. In the fifth example, the DTM splits the two water segments incorrectly (the two textures are very similar). The LDT substantially improves the segmentation, but the difficulties due to the great similarity of water patterns prove too difficult to overcome completely. More segmentation examples from this database are available in the supplemental [52].

Finally, the ocean-fire sequence from Chapter IV was segmented using TS-LDT, which was initialized with the DTM segmentation, and the results are shown in Figure V.8 for several frames. Similar to the LDT, the TS-LDT improves on the localization (tighter boundaries) and corrects the noise (spurious segments) of the DTM segmentation.
Figure V.6 LDT results on synthetic texture database: a) average Rand index versus the order of the motion model ($n$); and b) segmentation precision for the best $n$ for each algorithm. Each row presents the results for 2, 3, or 4 segments in the database.
Figure V.7 Results on the synthetic texture database: a) video; motion segmentations using b) DTM, and c) LDT. $r$ is the the Rand index of the segmentation.

Figure V.8 Segmentation of ocean-fire [15]: (a) video frames; and the segmentation using (b) DTM; and (c) TS-LDT.
V.H.3 Results on real video

We conclude the segmentation experiments with real video sequences. Figure V.9a presents the segmentation of a moving ferris wheel, using LDT and DTM for $K = \{2, 3\}$. For $K = 2$, both LDT and DTM segment the static background from the moving ferris wheel. However, for $K = 3$ regions, the plausible segmentation, by LDT, of the foreground into two regions corresponding to the ferris wheel and a balloon moving in the wind, is not matched by DTM. Instead, the latter segments the ferris wheel into two regions, according to the dominant direction of its local motion (either moving up or down), ignoring the balloon motion. This is identical to the problems found for the synthetic sequences of Figure V.5: the inability to uncover global homogeneity when the video is locally heterogeneous. On the other hand, the preference of LDT for two regions of very different sizes, illustrates its robustness to this problem. The strong local heterogeneity of the optical flow in the region of the ferris wheel is well explained by the global homogeneity of the corresponding layer dynamics. Figure V.9b shows another example of this phenomenon. For $K = 3$ regions, LDT segments the windmill into regions corresponding to the moving fan blades, parts of the shaking tail piece, and the background. When segmenting into $K = 4$ regions, LDT splits the fan blade segment into two regions, which correspond to the fan blades and the internal support pieces. On the other hand, the DTM segmentations for $K = \{3, 4\}$ split the
fan blades into different regions based on the orientation (vertical or horizontal) of the optical flow.

![Segmentation of a whirlpool using LDT with $K = \{2, 3, 4\}$.

We next illustrate an interesting property of LDT segmentation with the proposed initialization: that it tends to produce a sequence of segmentations which captures a hierarchy of scene dynamics. The whirlpool sequence of Figure V.10 (left) contains different levels of moving and turbulent water. For $K = 2$ layers, the LDT segments the scene into regions containing moving water and still background (still water and grass). Adding another layer splits the “moving water” segment into two regions of different water dynamics: slowly moving ripples (outside of the whirlpool) and fast turbulent water (inside the whirlpool). Finally for $K = 4$ layers, LDT splits the “turbulent water” region into two regions: the turbulent center of the whirlpool, and the fast water spiraling into it. Figure V.10 (right) shows the final segmentation, with the four layers corresponding to different levels of turbulence.

Finally, we present six other examples of LDT segmentation in Figure V.11. The first four are from the UCF database [96]. Figure V.11 (a-c) show segmentations of large pedestrian crowds. In Figure V.11a, a crowd moves in a circle around a pillar. The left side of the scene is less congested, and the crowd moves faster than on the right side. In Figure V.11b, the crowd moves with three levels of speed, which are stratified into horizontal layers. In Figure V.11c, a crowd gathers at the entrance of an escalator, with people moving quickly around the edges. These segmentations show that LDT can distinguish different speeds of crowd mo-
tion, regardless of the direction in which the crowd is traveling. In Figure V.11d, the LDT segments a highway scene into still background, the fast moving traffic on the highway, and the slow traffic that merges into it. Another whirlpool is shown in Figure V.11e, where the turbulent water component is segmented from the remaining moving water. Finally, Figure V.11f presents a windmill scene from [97], which the LDT segments into regions corresponding to the windmill (circular motion), the trees waving in the wind, and the static background. These examples demonstrate the robustness of the LDT representation, and its applicability to a wide range of scenes.

V.I Summary and discussion

In this chapter, we have introduced the layered dynamic texture, a generative model which represents video as a layered collection of dynamic textures of different appearance and dynamics. We have also derived the EM algorithm for estimation of the maximum-likelihood model parameters from training video sequences. Because the posterior distribution of layer assignments given observed video is computationally intractable, we have proposed two alternatives for inference with this model: a Gibbs sampler, and an efficient variational approximation.
The two approximate inference algorithms were compared experimentally, along with the corresponding approximate EM algorithms, on a synthetic dataset. The two approximations were shown to produce comparable marginals (and MAP segmentations) when the LDT is given, but the Gibbs sampler outperformed the variational approximation in the context of EM-based model learning. However, this improvement comes with a very significant computational cost. This trade-off between computation and performance is usually observed when there is a need to rely on approximate inference with these two methods.

We have also conducted extensive experiments, with both synthetic mosaics of real textures and real video sequences, that tested the ability of the proposed model (and algorithms) to segment video into regions of coherent dynamics and appearance. The combination of LDT and variational inference has been shown to outperform a number of state-of-the-art methods for video segmentation. In particular, it was shown to possess a unique ability to group regions of \textit{globally homogeneous but locally heterogeneous stochastic dynamics}. We believe that this ability is unmatched by any video segmentation algorithm currently available in the literature. The new method also has consistently produced segmentations with better spatial-localization than those possible with the \textit{localized representations}, such as the mixture of dynamic textures, that have previously been prevalent in the area of dynamic texture segmentation. Finally, we have demonstrated the robustness of the model, by segmenting real video sequences depicting different classes of scenes: various types of crowds, highway traffic, and scenes containing a combination of globally homogeneous motion and highly stochastic motion (e.g. rotating windmills plus waving tree branches, or whirlpools).

\section{V.J Acknowledgements}

The authors thank Rene Vidal for the code from [17, 19], Mubarak Shah for the crowd videos [88, 96], and Renaud Péteri, Mark Huiskes and Sándor Fazekas
for the windmill video [97]


V.K Appendix

V.K.1 Derivations for the LDT

In this appendix, we present the derivations of the M-step of the EM algorithm and the variational approximation for the LDT. In addition, we also derive an efficient algorithm for sampling a state-sequence from an LDS.

Derivation of the M-Step for LDT

The maximization of the \( Q \) function with respect to the LDT parameters leads to two optimization problems. The first is a maximization with respect to a square matrix \( X \) of the same form as (IV.37),

\[
X^* = \arg\max_X -\frac{1}{2}\text{tr} \left( X^{-1}A \right) - \frac{b}{2} \log |X| \Rightarrow X^* = \frac{1}{b} A. \quad (V.71)
\]

The second is a maximization with respect to a matrix \( X \) with the same form as (IV.42),

\[
X^* = \arg\max_X -\frac{1}{2}\text{tr} \left[ D(-BX^T - XB^T + XCX^T) \right] \Rightarrow X^* = BC^{-1}, \quad (V.72)
\]

where \( D \) and \( C \) are symmetric and invertible matrices.
The optimal parameters are found by collecting the relevant terms in (V.20) and maximizing. This leads to a number of problems of the form of (V.72), namely

\[
A^{(j)*} = \arg\max_{A^{(j)}} -\frac{1}{2} \text{tr} \left( Q^{(j)-1} \left( (\bar{A}^{(j)}(\bar{A}^{(j)})^T + A^{(j)}(A^{(j)})^T) = (\bar{A}^{(j)}(\bar{A}^{(j)})^T + A^{(j)}(A^{(j)})^T) \right) \right),
\]

\[
\mu^{(j)*} = \arg\max_{\mu^{(j)}} -\frac{1}{2} \text{tr} \left( Q^{(j)-1} \left( -x_1(\mu^{(j)})^T - \mu^{(j)}(x_1(\mu^{(j)})^T + \mu^{(j)}(\mu^{(j)})^T) \right) \right),
\]

\[
C_i^{(j)*} = \arg\max_{C_i^{(j)}} -\frac{1}{2} \frac{1}{r^{(j)}} \bar{z}_i^{(j)} \left( -2C_i^{(j)}(C_i^{(j)} + C_i^{(j)}(\Phi_i^{(j)} C_i^{(j)})) \right).
\]

Using (V.72) leads to the solutions of (V.12). The remaining problems are of the form of (V.71)

\[
Q^{(j)*} = \arg\max_{Q^{(j)}} -\frac{1}{2} \text{tr}Q^{(j)-1} \left( \hat{P}^{(j)} - x_1(\mu^{(j)})^T - \mu^{(j)}(x_1(\mu^{(j)})^T + \mu^{(j)}(\mu^{(j)})^T) + \phi_2^{(j)} - \psi^{(j)}(\mu^{(j)}(\phi_1^{(j)} A^{(j)})) - \frac{\tau}{2} \log |Q^{(j)}| ,
\]

\[
r^{(j)*} = \arg\max_{r^{(j)}} -\frac{1}{2} \frac{1}{r^{(j)}} \sum_{i=1}^m \bar{z}_i^{(j)} \left( \sum_{t=1}^\tau y_{i,t}^2 - 2C_i^{(j)}(\Gamma_i^{(j)} + C_i^{(j)}(\Phi_i^{(j)} C_i^{(j)})) \right) \right) - \frac{\tau}{2} \hat{N}_j \log r^{(j)}.
\]

In the first case, it follows from (V.71) that

\[
Q^{(j)*} = \frac{1}{\tau} \left( \hat{P}^{(j)} - x_1(\mu^{(j)})^T - \mu^{(j)}(x_1(\mu^{(j)})^T + \mu^{(j)}(\mu^{(j)})^T) + \phi_2^{(j)} - \psi^{(j)}(\mu^{(j)}(\phi_1^{(j)} A^{(j)})) \right)
\]

\[
= \frac{1}{\tau} \left( \hat{P}^{(j)} - \mu^{(j)*}(\mu^{(j)*} + \phi_2^{(j)} - \psi^{(j)}(\mu^{(j)}(\phi_1^{(j)} A^{(j)})) \right).
\]

In the second case,

\[
r^{(j)*} = \frac{1}{\tau \hat{N}_j} \sum_{i=1}^m \bar{z}_i^{(j)} \left( \sum_{t=1}^\tau y_{i,t}^2 - 2C_i^{(j)}(\Gamma_i^{(j)} + C_i^{(j)}(\Phi_i^{(j)} C_i^{(j)})) \right) \right)
\]

\[
= \frac{1}{\tau \hat{N}_j} \sum_{i=1}^m \bar{z}_i^{(j)} \left( \sum_{t=1}^\tau y_{i,t}^2 - C_i^{(j)*}(\Gamma_i^{(j)} \right).
\]

Sampling a state sequence from an LDS conditioned on the observation $\mathcal{X}_1 = \{x_1, \cdots, x_\tau\}$ from an LDS with parameters $\Theta = \{A, Q, R, \mu, S\}$, conditioned on
the observed sequence \( y_{1:\tau} = \{ y_1, \cdots, y_\tau \} \). The sampling algorithm first runs the Kalman filter (Section II.C) to compute state estimates conditioned on the current observations

\[
\hat{x}_{t|t-1} = \mathbb{E}(x_t|y_{1:t-1}), \quad \hat{V}_{t|t-1} = \text{cov}(x_t|y_{1:t-1}),
\]
\[
\hat{x}_{t|t} = \mathbb{E}(x_t|y_{1:t}), \quad \hat{V}_{t|t} = \text{cov}(x_t|y_{1:t}).
\] (V.77)

From the Markovian structure of the LDS (Figure II.1), \( p(x_{1:\tau}|y_{1:\tau}) \) can be factored in reverse-order

\[
p(x_{1:\tau}|y_{1:\tau}) = p(x_{\tau}|y_{1:\tau}) \prod_{t=1}^{\tau-1} p(x_t|x_{t+1}, y_{1:\tau})
\] (V.78)
\[
= p(x_{\tau}|y_{1:\tau}) \prod_{t=1}^{\tau-1} p(x_t|x_{t+1}, y_{1:t}).
\] (V.79)

\( p(x_{\tau}|y_{1:\tau}) \) is a Gaussian with parameters already computed by the Kalman filter, i.e. \( x_{\tau} \sim \mathcal{N}(\hat{x}_{\tau|\tau}, \hat{V}_{\tau|\tau}) \). The remaining distributions, \( p(x_t|x_{t+1}, y_{1:t}) \), are Gaussian with mean and covariance given by the conditional Gaussian theorem [24],

\[
\mu_t = \mathbb{E}[x_t|x_{t+1}, y_{1:t}] \quad \text{ (V.80)}
\]
\[
= \mathbb{E}[x_t|y_{1:t}] + \text{cov}(x_t, x_{t+1}|y_{1:t})\text{cov}(x_{t+1}|y_{1:t})^{-1}(x_{t+1} - \mathbb{E}[x_{t+1}|y_{1:t}])
\]
\[
= \hat{x}_{t|t} + \hat{V}_{t|t} A^T (\hat{V}_{t+1|t})^{-1} (x_{t+1} - \hat{x}_{t+1|t}),
\] (V.81)
\[
\Sigma_t = \text{cov}(x_t|x_{t+1}, y_{1:t}) \quad \text{ (V.82)}
\]
\[
= \text{cov}(x_t|y_{1:t}) - \text{cov}(x_t, x_{t+1}|y_{1:t})\text{cov}(x_{t+1}|y_{1:t})^{-1}\text{cov}(x_{t+1}, x_t|y_{1:t})
\]
\[
= \hat{V}_{t|t} - \hat{V}_{t|t} A^T (\hat{V}_{t+1|t})^{-1} A \hat{V}_{t|t},
\] (V.83)

where we have used \( \text{cov}(x_t, x_{t+1}|y_{1:t}) = \text{cov}(x_t, Ax_t|y_{1:t}) = \hat{V}_{t|t} A^T \). A state sequence \( (x_1, \cdots, x_\tau) \) can thus be sampled in reverse order, with \( x_\tau \sim \mathcal{N}(\hat{x}_{\tau|\tau}, \hat{V}_{\tau|\tau}) \) and \( x_t \sim \mathcal{N}(\mu_t, \Sigma_t) \) for \( 0 < t < \tau \).

### Derivation of the variational approximation for LDT

In this appendix, we derive a variational approximation for the LDT. The \( \mathcal{L} \) function of (V.35) is minimized by sequentially optimizing each of the factors
\( q(x^{(j)}) \) and \( q(z_i) \), while holding the remaining constant \([93]\). For convenience, we define the variable \( W = \{X, Z\} \). Rewriting (V.35) in terms of a single factor \( q(w_l) \), while holding all others constant,

\[
\mathcal{L}(q(W)) 
\propto \int q(w_l) \log q(w_l) dw_l - \int q(w_l) \left( \prod_{k \neq l} q(w_k) \log p(W,Y) dW \right) \tag{V.84}
\]

\[
= \int q(w_l) \log q(w_l) dw_l - \int q(w_l) \log \tilde{p}(w_l,Y) dw_l \tag{V.85}
\]

\[
= D(q(w_l) \| \tilde{p}(w_l,Y)) \tag{V.86}
\]

where in (V.84) we have dropped terms that do not depend on \( q(w_l) \) (and hence do not affect the optimization), and defined \( \tilde{p}(w_l,Y) \) as

\[
\log \tilde{p}(w_l,Y) \propto \mathbb{E}_{W_{k \neq l}}[\log p(W,Y)], \tag{V.87}
\]

where \( \mathbb{E}_{W_{k \neq l}}[\log p(W,Y)] = \int \prod_{k \neq l} q(w_k) \log p(W,Y) dW_{k \neq l} \). Since (V.86) is minimized when \( q^*(w_l) = \tilde{p}(w_l,Y) \), the optimal factor \( q(w_l) \) is equal to the expectation of the joint log-likelihood with respect to the other factors \( W_{k \neq l} \). We next derive the forms of the optimal factors \( q(x^{(j)}) \) and \( q(z_i) \). For convenience, we ignore normalization constants during the derivation, and reinstate them after the forms of the factors are known.

**Optimization of \( q(x^{(j)}) \)**

Rewriting (V.87) with \( w_l = x^{(j)} \),

\[
\log q^*(x^{(j)}) \propto \log \tilde{p}(x^{(j)},Y) = \mathbb{E}_{Z,X_{k \neq j}}[\log p(X,Y,Z)] \tag{V.88}
\]

\[
\propto \mathbb{E}_{Z,X_{k \neq j}} \left[ \sum_{i=1}^{m} z_i^{(j)} \log p(y_i|x^{(j)}, z_i = j) + \log p(x^{(j)}) \right] \tag{V.89}
\]

\[
= \sum_{i=1}^{m} \mathbb{E}_{z_i}[z_i^{(j)}] \log p(y_i|x^{(j)}, z_i = j) + \log p(x^{(j)}), \tag{V.90}
\]

where (V.89) drops the terms of the complete data log-likelihood (V.13) that are not a function of \( x^{(j)} \). Finally, defining \( h_i^{(j)} = \mathbb{E}_{z_i}[z_i^{(j)}] = \int q(z_i) z_i^{(j)} dz_i \), and the
normalization constant

\[ Z_q^{(j)} = \int p(x^{(j)}) \prod_{i=1}^{m} p(y_i|x^{(j)}, z_i = j) h_i^{(j)} \, dx^{(j)}, \quad (V.91) \]

the optimal \( q(x^{(j)}) \) is given by (V.36).

**Optimization of \( q(z_i) \)**

Rewriting (V.87) with \( w_i = z_i \) and dropping terms that do not depend on \( z_i \),

\[
\log q^*(z_i) \propto \log \tilde{p}(z_i, Y) = \mathbb{E}_{X, Z_k \neq i} [\log p(X, Y, Z)]
\]

\[
\propto \mathbb{E}_{X, Z_k \neq i} \left[ \sum_{j=1}^{K} z_i^{(j)} \log p(y_i|x^{(j)}, z_i = j) + \log p(Z) \right]
\]

\[
= \sum_{j=1}^{K} z_i^{(j)} \mathbb{E}_{x^{(j)}} [\log p(y_i|x^{(j)}, z_i = j)] + \mathbb{E}_{Z_k \neq i} [\log p(Z)]. \quad (V.93)
\]

For the last term, we have

\[
\mathbb{E}_{Z_k \neq i} [\log p(Z)] \propto \mathbb{E}_{Z_{k \neq i}} [\log (V_i(z_i) \prod_{(i',i') \in E} V_{i,i'}(z_i, z_{i'}))]
\]

\[
= \log V_i(z_i) + \sum_{(i',i') \in E} \mathbb{E}_{z_{i'}} [\log V_{i,i'}(z_i, z_{i'})] \quad (V.94)
\]

\[
= \sum_{j=1}^{K} z_i^{(j)} \log \alpha_i^{(j)} + \sum_{(i',i') \in E} \mathbb{E}_{z_{i'}} [\sum_{j=1}^{K} z_i^{(j)} z_{i'}^{(j)} \log \frac{\gamma_1}{\gamma_2} + \log \gamma_2] \quad (V.96)
\]

\[
\propto \sum_{j=1}^{K} z_i^{(j)} \log \alpha_i^{(j)} + \sum_{j=1}^{K} z_i^{(j)} \sum_{(i',i') \in E} \mathbb{E}_{z_{i'}} [z_{i'}^{(j)}] \log \frac{\gamma_1}{\gamma_2} \quad (V.97)
\]

\[
= \sum_{j=1}^{K} z_i^{(j)} \left( \log \alpha_i^{(j)} + \sum_{(i',i') \in E} h_{i'}^{(j)} \log \frac{\gamma_1}{\gamma_2} \right). \quad (V.98)
\]

Hence,

\[
\log q^*(z_i) \propto \sum_{j=1}^{K} z_i^{(j)} \left( \mathbb{E}_{x^{(j)}} [\log p(y_i|x^{(j)}, z_i = j)] + \sum_{(i',i') \in E} h_{i'}^{(j)} \log \frac{\gamma_1}{\gamma_2} + \log \alpha_i^{(j)} \right)
\]

\[
= \sum_{j=1}^{K} z_i^{(j)} \log (g_i^{(j)} \alpha_i^{(j)}), \quad (V.99)
\]
where \( g_i^{(j)} \) is defined in (V.39). This is a multinomial distribution of normalization constant \( \sum_{j=1}^{K} \alpha_i^{(j)} g_i^{(j)} \), leading to (V.37) with \( h_i^{(j)} \) as given in (V.38).

**Normalization constant for** \( q(x^{(j)}) \)

Taking the log of (V.91),

\[
\log \mathcal{Z}_q^{(j)} = \log \int p(x^{(j)}) \prod_{i=1}^{m} p(y_i|x^{(j)}, z_i = j) h_i^{(j)} dx^{(j)}
\]

\[
= \log \int p(x^{(j)}) \prod_{i=1}^{m} \prod_{t=1}^{\tau} p(y_{i,t}|x_t^{(j)}, z_i = j) h_i^{(j)} dx^{(j)}.
\]

Note that the term \( p(y_{i,t}|x_t^{(j)}, z_i = j) h_i^{(j)} \) does not affect the integral when \( h_i^{(j)} = 0 \).

Defining \( I_j \) as the set of indices with non-zero \( h_i^{(j)} \), i.e. \( I_j = \{i | h_i^{(j)} > 0\} \), (V.101) becomes

\[
\log \mathcal{Z}_q^{(j)} = \log \int p(x^{(j)}) \prod_{i \in I_j} \prod_{t=1}^{\tau} p(y_{i,t}|x_t^{(j)}, z_i = j) h_i^{(j)} dx^{(j)},
\]

where

\[
p(y_{i,t}|x_t^{(j)}, z_i = j) h_i^{(j)} = G(y_{i,t}, \tilde{C}_i^{(j)} x_t^{(j)}, \tilde{r}^{(j)}) h_i^{(j)}
\]

\[
= (2\pi r^{(j)})^{-\frac{1}{2} h_i^{(j)}} \left( \frac{2\pi r^{(j)}}{h_i^{(j)}} \right)^{\frac{1}{2}} G \left( y_{i,t}, \tilde{C}_i^{(j)} x_t^{(j)}, \tilde{r}^{(j)} h_i^{(j)} \right).
\]

For convenience, we define an LDS over the subset \( I_j \) parameterized by \( \tilde{\Theta}_j = \{A^{(j)}, Q^{(j)}, \tilde{C}^{(j)}, \tilde{R}, \tilde{\mu}^{(j)}\} \), where \( \tilde{C}^{(j)} = [C_i^{(j)}]_{i \in I_j} \), and \( \tilde{R} \) is diagonal with entries \( \tilde{r}_i^{(j)} = \frac{r_i^{(j)}}{h_i^{(j)}} \) for \( i \in I_j \). Noting that this LDS has conditional observation likelihood

\[
\tilde{p}(y_{i,t}|x_t^{(j)}, z_i = j) = G(y_{i,t}, \tilde{C}_i^{(j)} x_t^{(j)}, \tilde{r}_i^{(j)}),
\]

we can rewrite \( p(y_{i,t}|x_t^{(j)}, z_i = j) h_i^{(j)} = (2\pi r^{(j)})^{-\frac{1}{2}(1-h_i^{(j)})} (h_i^{(j)})^{-\frac{1}{2}} \tilde{p}(y_{i,t}|x_t^{(j)}, z_i = j) \) and, from (V.102),

\[
\log \mathcal{Z}_q^{(j)} =
\]

\[
\log \int p(x^{(j)}) \prod_{i \in I_j} \prod_{t=1}^{\tau} \left[ (2\pi r^{(j)})^{-\frac{1}{2}(1-h_i^{(j)})} (h_i^{(j)})^{-\frac{1}{2}} \tilde{p}(y_{i,t}|x_t^{(j)}, z_i = j) \right] dx^{(j)}.
\]

Since, under the restricted LDS, the likelihood of \( Y_j = [y_i]_{i \in I_j} \) is

\[
\tilde{p}_j(Y_j) = \int p(x^{(j)}) \prod_{i \in I_j} \prod_{t=1}^{\tau} \tilde{p}(y_{i,t}|x_t^{(j)}, z_i = j) dx^{(j)},
\]

(V.106)
it follows that
\[
\log Z_{q_j}^{(j)} = \log \left[ \tilde{p}_j(Y_j) \prod_{i \in I_j} \prod_{t=1}^{\tau} (2\pi r_t^{(j)})^{\frac{1}{2} (1 - h_i^{(j)}) (h_i^{(j)})^{-\frac{1}{2}}} \right] \quad (V.107)
\]
\[
= \frac{\tau}{2} \sum_{i \in I_j} (1 - h_i^{(j)}) \log (2\pi r_t^{(j)}) - \frac{\tau}{2} \sum_{i \in I_j} \log h_i^{(j)} + \log \tilde{p}_j(Y_j). \quad (V.108)
\]

**Lower bound on \( p(Y) \)**

To lower bound \( p(Y) \) as in (V.45), we compute the \( \mathcal{L} \) function of (V.32). We start with
\[
\log \frac{q(X, Z)}{p(X, Y, Z)} = \log q(X, Z) - \log p(X, Y, Z) \quad (V.109)
\]
\[
= \left[ \sum_j \log q(x_j^{(j)}) + \sum_i \log q(z_i) \right] \quad (V.110)
\]
\[
- \left[ \sum_{j,i} z_i^{(j)} \log p(y_i|x_j^{(j)}, z_i = j) + \sum_j \log p(x_j^{(j)}) + \log p(Z) \right].
\]

Substituting the optimal \( q^* \) of (V.36) and (V.37),
\[
\log \frac{q(X, Z)}{p(X, Y, Z)} = \left[ \sum_{j,i} h_i^{(j)} \log p(y_i|x_j^{(j)}, z_i = j) + \sum_j \log p(x_j^{(j)}) \right. \\
- \left. \sum_j \log Z_q^{(j)} + \sum_{j,i} z_i^{(j)} \log h_i^{(j)} \right] - \left[ \sum_{j,i} z_i^{(j)} \log p(y_i|x_j^{(j)}, z_i = j) \right. \\
+ \sum_j \log p(x_j^{(j)}) + \log p(Z) \right]
\]
\[
= \sum_{j,i} (h_i^{(j)} - z_i^{(j)}) \log p(y_i|x_j^{(j)}, z_i = j) - \sum_j \log Z_q^{(j)} \\
+ \sum_{j,i} z_i^{(j)} \log h_i^{(j)} - \log p(Z). \quad (V.112)
\]

Noting that
\[
\log p(Z) = \sum_{j,i} z_i^{(j)} \log \alpha_i^{(j)} + \sum_{(i,i') \in E} \left( \log \gamma_2 + \sum_j z_i^{(j)} z_i^{(j)} \log \frac{\gamma_i}{\gamma_2} \right) - \log Z_Z,
\]
we have
\[
\log \frac{q(X, Z)}{p(X, Y, Z)} = \sum_{j,i} (h_{i}^{(j)} - z_{i}^{(j)}) \log p(y_{i}^{(j)}, z_{i} = j) - \sum_{j} \log Z_{q}^{(j)} 
\]
\[
+ \sum_{j,i} z_{i}^{(j)} \log \frac{h_{i}^{(j)}}{\alpha_{i}^{(j)}} - \sum_{(i,i') \in E} \left( \log \gamma_{2} + \sum_{j} z_{i}^{(j)} z_{i'}^{(j)} \log \frac{\gamma_{1}}{\gamma_{2}} \right) + \log Z_{Z}.
\]

Taking the expectation of (V.113) with respect to \( q^{*}(X, Z) \) yields the KL divergence,
\[
D(q(X, Z) \parallel p(X, Y, Z)) = -\sum_{j} \log Z_{q}^{(j)} 
\]
\[
+ \sum_{j,i} h_{i}^{(j)} \log \frac{h_{i}^{(j)}}{\alpha_{i}^{(j)}} - \sum_{(i,i') \in E} \left( \log \gamma_{2} + \sum_{j} h_{i}^{(j)} h_{i'}^{(j)} \log \frac{\gamma_{1}}{\gamma_{2}} \right) + \log Z_{Z}.
\]

Substituting into (V.45) yields the log-likelihood lower-bound (V.46).

V.K.2 Derivations for the TS-LDT

In this section, we derive the EM algorithm and the variational approximation for the TS-LDT. The derivations follow closely to those of the standard LDT.

Derivation of the EM algorithm for TS-LDT

Complete data log-likelihood: Taking the logarithm of (V.53), the complete data log-likelihood is
\[
\ell(X, Y, Z) = \sum_{i=1}^{m} \sum_{j=1}^{K} \sum_{t=1}^{T} z_{i,t}^{(j)} \log p(y_{i,t}^{(j)}, z_{i,t} = j) 
\]
\[
+ \sum_{j=1}^{K} \left( \log p(x_{1}^{(j)}) + \sum_{t=2}^{T} \log p(x_{t}^{(j)} | x_{t-1}^{(j)}) \right) + \log p(Z).
\]
Using (V.5) and (V.54) and dropping terms that do not depend on the parameters \( \Theta \) (and thus play no role in the M-step),

\[
\ell(X, Y, Z) = -\frac{1}{2} \sum_{j=1}^{K} \sum_{i=1}^{m} \sum_{t=1}^{\tau} z_{i,t}^{(j)} \left( \| y_{i,t} - C_i^{(j)} x_t^{(j)} - \gamma_i^{(j)} \|_{r^{(j)}}^2 + \log r^{(j)} \right) + \frac{1}{2} \sum_{j=1}^{K} \left( \| x_1^{(j)} - \mu_j \|_{Q^{(j)}}^2 + \sum_{t=2}^{\tau} \| x_t^{(j)} - A^{(j)} x_{t-1}^{(j)} \|_{Q^{(j)}}^2 + \tau \log |Q^{(j)}| \right).
\]

Note that \( p(Z) \) can be ignored since the parameters of the MRF are constants.

Finally, the complete data log-likelihood is

\[
\ell(X, Y, Z) = -\frac{1}{2} \sum_{j=1}^{K} \sum_{i=1}^{m} \sum_{t=1}^{\tau} z_{i,t}^{(j)} \left( (y_{i,t} - \gamma_i^{(j)})^2 - 2(y_{i,t} - \gamma_i^{(j)}) C_i^{(j)} x_t^{(j)} \right)
+ C_i^{(j)} P_{t,t}^{(j)} C_i^{(j)T}
- \frac{1}{2} \sum_{j=1}^{K} \sum_{t=2}^{\tau} \text{tr} \left( Q^{(j)-1} \left( P_{t,t}^{(j)} - P_{t,t-1}^{(j)} A^{(j)T} - A^{(j)} P_{t,t-1}^{(j)T} + A^{(j)T} P_{t-1,t-1}^{(j)} A^{(j)} \right) \right)
- \frac{1}{2} \sum_{j=1}^{K} \sum_{i=1}^{m} \sum_{t=1}^{\tau} z_{i,t}^{(j)} \log r^{(j)} - \frac{\tau}{2} \sum_{j=1}^{K} \log |Q^{(j)}|,
\]

where we define \( P_{t,t}^{(j)} = x_t^{(j)} x_t^{(j)T} \) and \( P_{t,t-1}^{(j)} = x_t^{(j)} x_{t-1}^{(j)T} \).

**E-step:** From (V.117), it follows that the E-step of (V.11) requires conditional expectations of two forms:

\[
E_{X,Z|Y}[f(x^{(j)})] = E_{X|Y}[f(x^{(j)})], \tag{V.118}
\]
\[
E_{X,Z|Y}[z_{i,t}^{(j)} f(x^{(j)})] = E_{Z|Y}[z_{i,t}^{(j)}] E_{X|z_{i,t}=j}[f(x^{(j)})]. \tag{V.119}
\]

for some function \( f \) of \( x^{(j)} \), and where \( E_{X|Y,z_{i,t}=j} \) is the conditional expectation of \( X \) given the observation \( Y \) and that the i-th pixel at time \( t \) belongs to layer \( j \).
substituting (V.117) into (V.11), leads to the \( Q \) function

\[
Q(\Theta; \hat{\Theta}) = -\frac{1}{2} \sum_{j=1}^{K} \frac{1}{r^{(j)}} \sum_{i=1}^{m} \left( \sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)} (y_{i,t} - \gamma_{i}^{(j)})^2 - 2C_{i}^{(j)} \Gamma_{i}^{(j)} + C_{i}^{(j)} \Phi_{i}^{(j)} C_{i}^{(j)T} \right) \\
- \frac{1}{2} \sum_{j=1}^{K} \text{tr} \left( Q^{(j)-1} \left( \hat{P}_{1,j}^{(j)} - \hat{x}_{1}^{(j)} \mu_{j}^{(j)} - \mu_{j}^{(j)} (\hat{x}_{1}^{(j)})^T + \mu_{j}^{(j)} \mu_{j}^{(j)T} + \phi_{j}^{2} - \psi_{j}A_{j}^{T} \\
- A_{j}^{(j)\psi^{(j)T}} + A_{j}^{(j)\phi_{j}^{(j)} A_{j}^{(j)T})} \right) \right) - \frac{1}{2} \sum_{j=1}^{K} \tilde{N}_{j} \log r^{(j)} - \frac{1}{2} \sum_{j=1}^{K} \log |Q^{(j)}| \quad \text{(V.120)}
\]

**M-step**: Maximizing the \( Q \) function, the M-step for the TS-LDT is similar to that of the LDT, except for the estimates of \( r^{(j)\ast} \) and \( \gamma_{i}^{(j)} \). For the former,

\[
r^{(j)\ast} = \frac{1}{N_{j}} \sum_{i=1}^{m} \left[ \sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)} (y_{i,t} - \gamma_{i}^{(j)})^2 - 2C_{i}^{(j)} \Gamma_{i}^{(j)} + C_{i}^{(j)} \Phi_{i}^{(j)} C_{i}^{(j)T} \right] \quad \text{(V.121)}
\]

\[
= \frac{1}{N_{j}} \sum_{i=1}^{m} \left[ \sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)} (y_{i,t} - \gamma_{i}^{(j)})^2 - C_{i}^{(j)\ast} \Gamma_{i}^{(j)} \right]. \quad \text{(V.122)}
\]

For the latter, noting that \( \frac{\partial}{\partial \gamma_{i}^{(j)}} \Gamma_{i}^{(j)} = - \sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)} \hat{z}_{i,t}^{(j)} = - \xi_{i}^{(j)} \),

\[
\frac{\partial Q}{\partial \gamma_{i}^{(j)}} = \frac{1}{r^{(j)}} \left( \sum_{t=1}^{\tau} -2\hat{z}_{i,t}^{(j)} (y_{i,t} - \gamma_{i}^{(j)}) + 2C_{i}^{(j)} \xi_{i}^{(j)} \right) = 0, \quad \text{(V.123)}
\]

\[
\Rightarrow \gamma_{i}^{(j)} = \frac{1}{\sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)}} \left( \sum_{t=1}^{\tau} \hat{z}_{i,t}^{(j)} y_{i,t} - C_{i}^{(j)} \xi_{i}^{(j)} \right). \quad \text{(V.124)}
\]

**Derivation of the variational approximation for TS-LDT**

Substituting (V.60) into (V.32), leads to

\[
\mathcal{L}(q(X, Z)) = \int \prod_{j} q(x^{(j)}) \prod_{i,t} q(z_{i,t}) \log \frac{\prod_{j} q(x^{(j)}) \prod_{i,t} q(z_{i,t})}{p(X, Y, Z)} dX dZ. \quad \text{(V.125)}
\]

The \( \mathcal{L} \) function (V.125) is minimized by sequentially optimizing each of the factors \( q(x^{(j)}) \) and \( q(z_{i,t}) \), while holding the others constant \([93]\).
Optimization of $q(x^{(j)})$: Rewriting (V.87) with $w_l = x^{(j)}$,

$$\log q^*(x^{(j)}) \propto \log \bar{p}(x^{(j)}, Y) = \mathbb{E}_{Z, X_{k \neq j}}[\log p(X, Y, Z)] \quad (V.126)$$

$$\propto \mathbb{E}_{Z, X_{k \neq j}} \left[ \sum_{t=1}^{\tau} \sum_{i=1}^{m} z_{i,t}^{(j)} \log p(y_{i,t}|x_t^{(j)}, z_{i,t} = j) + \log p(x^{(j)}) \right] \quad (V.127)$$

$$= \sum_{t=1}^{\tau} \sum_{i=1}^{m} \mathbb{E}_{z_{i,t}}[z_{i,t}^{(j)}] \log p(y_{i,t}|x_t^{(j)}, z_{i,t} = j) + \log p(x^{(j)}), \quad (V.128)$$

where in (V.127) we have dropped the terms of the complete data log-likelihood (V.115) that are not a function of $x^{(j)}$. Finally, defining $h_{i,t}^{(j)} = \mathbb{E}_{z_{i,t}}[z_{i,t}^{(j)}] = \int q(z_{i,t}) z_{i,t}^{(j)} \, dz_{i,t}$, and the normalization constant

$$Z_q^{(j)} = \int p(x^{(j)}) \prod_{t=1}^{\tau} \prod_{i=1}^{m} p(y_{i,t}|x_t^{(j)}, z_{i,t} = j)^{h_{i,t}^{(j)}} \, dx^{(j)}, \quad (V.129)$$

the optimal $q(x^{(j)})$ is given by (V.61).

Optimization of $q(z_{i,t})$: Rewriting (V.87) with $w_l = z_{i,t}$ and dropping terms that do not depend on $z_{i,t}$,

$$\log q^*(z_{i,t}) \propto \log \bar{p}(z_{i,t}, Y) = \mathbb{E}_{X, Z_{k \neq i,s \neq t}}[\log p(X, Y, Z)] \quad (V.130)$$

$$\propto \mathbb{E}_{X, Z_{k \neq i,s \neq t}} \left[ \sum_{j=1}^{K} z_{i,t}^{(j)} \log p(y_{i,t}|x_t^{(j)}, z_{i,t} = j) + \log p(Z) \right] \quad (V.131)$$

$$= \sum_{j=1}^{K} z_{i,t}^{(j)} \mathbb{E}_{x_t^{(j)}}[\log p(y_{i,t}|x_t^{(j)}, z_{i,t} = j)] + \mathbb{E}_{Z_{k \neq i,s \neq t}}[\log p(Z)]. \quad (V.132)$$
For the last term,

\[
\mathbb{E}_{z_{k\neq i,s\neq t}}[\log p(Z)]
\]

(V.133)

\[
\propto \mathbb{E}_{z_{k\neq i,s\neq t}}[\log(V_{i,t}(z_{i,t})) \prod_{(i',t') \in E} V_{i',t'}(z_{i,t}, z_{i',t'})]
\]

(V.134)

\[
= \log V_{i,t}(z_{i,t}) + \sum_{(i',t') \in E} \mathbb{E}_{z_{i',t'}}[\log V_{i',t'}(z_{i,t}, z_{i',t'})]
\]

(V.135)

\[
= \sum_{j=1}^{K} z_{i,t}^{(j)} \log \alpha_{i,t}^{(j)} + \sum_{(i',t') \in E} \sum_{j=1}^{K} z_{i,t}^{(j)} z_{i',t'}^{(j)} \log \frac{\gamma_1}{\gamma_2} + \log \gamma_2
\]

(V.136)

\[
= \prod_{j=1}^{K} \left[ \log \alpha_{i,t}^{(j)} + \sum_{(i',t') \in E} h_{i',t'}^{(j)} \log \frac{\gamma_1}{\gamma_2} + \sum_{(i',t') \in E} h_{i',t'}^{(j)} \log \frac{\beta_1}{\beta_2} \right].
\]

(V.137)

Hence,

\[
\log q^*(z_{i,t}) \propto \sum_{j=1}^{K} z_{i,t}^{(j)} \left( \mathbb{E}_{x_{t}^{(j)}}[\log p(y_{i,t}|x_{t}^{(j)}, z_{i,t} = j)] + \sum_{(i',t') \in E} h_{i',t'}^{(j)} \log \frac{\gamma_1}{\gamma_2} \right)
\]

(V.138)

where \(q_{i,t}^{(j)}\) is defined in (V.64). This is a multinomial distribution of normalization constant \(\sum_{j=1}^{K} (\alpha_{i,t}^{(j)} q_{i,t}^{(j)})\), leading to (V.62) with \(h_{i,t}^{(j)}\) as given in (V.63).

**Normalization constant for \(q(x^{(j)})\):** Taking the log of (V.129)

\[
\log Z_{q}^{(j)} = \log \int p(x^{(j)}) \prod_{i=1}^{m} \prod_{t=1}^{\tau} p(y_{i,t}|x_{t}^{(j)}, z_{i,t} = j)^{h_{i,t}^{(j)}} dx^{(j)}.
\]

(V.139)
Note that the term \( p(y_{i,t} | x_t^{(j)}, z_{i,t} = j)^{h_{i,t}^{(j)}} \) does not affect the integral when \( h_{i,t}^{(j)} = 0 \).

Defining \( \mathcal{I}_j \) as the set of indices \((i, t)\) with non-zero \( h_{i,t}^{(j)} \), i.e. \( \mathcal{I}_j = \{(i, t) | h_{i,t}^{(j)} > 0\} \), (V.139) becomes

\[
\log Z_q^{(j)} = \log \int p(x^{(j)}) \prod_{(i,t) \in \mathcal{I}_j} p(y_{i,t} | x_t^{(j)}, z_{i,t} = j)^{h_{i,t}^{(j)}} dx^{(j)}, \tag{V.140}
\]

where

\[
p(y_{i,t} | x_t^{(j)}, z_{i,t} = j)^{h_{i,t}^{(j)}} = G(y_{i,t}, C_i^{(j)} x_t^{(j)}, r^{(j)})^{h_{i,t}^{(j)}} \tag{V.141}
= (2\pi r^{(j)})^{-\frac{1}{2} h_{i,t}^{(j)}} \left( \frac{2\pi r^{(j)}}{h_{i,t}^{(j)}} \right)^{\frac{1}{2}} G(y_{i,t}, C_i^{(j)} x_t^{(j)}, r^{(j)})^{h_{i,t}^{(j)}}. \tag{V.142}
\]

For convenience, we define an LDS over the subset of observations indexed by \( \mathcal{I}_j \). Note that the dimension of the observation \( y_t \) changes over time, depending on how many \( h_{i,t}^{(j)} \) are active in each frame. The LDS is parameterized by \( \bar{\Theta}_j = \{A^{(j)}, Q, \bar{C}_t^{(j)}, \bar{R}_t^{(j)}, \mu^{(j)}\} \), where \( \bar{C}_t^{(j)} = [C_i^{(j)}]_{(i,t) \in \mathcal{I}_j} \) is a time-varying observation matrix, and \( \bar{R}_t^{(j)} \) is time-varying diagonal covariance matrix with diagonal entries \( \left[ \frac{r^{(j)}}{h_{i,t}^{(j)}} \right]_{(i,t) \in \mathcal{I}_j} \). This LDS has conditional observation likelihood \( \tilde{p}(y_{i,t} | x_t^{(j)}, z_{i,t} = j) = G(y_{i,t}, C_i^{(j)} x_t^{(j)}, \bar{R}_t^{(j)}) \), we can rewrite

\[
p(y_{i,t} | x_t^{(j)}, z_{i,t} = j)^{h_{i,t}^{(j)}} = (2\pi r^{(j)})^{\frac{1}{2}} (1 - h_{i,t}^{(j)})^{\frac{1}{2}} (h_{i,t}^{(j)})^{\frac{1}{2}} \tilde{p}(y_{i,t} | x_t^{(j)}, z_{i,t} = j), \tag{V.143}
\]

and, from (V.140),

\[
\log Z_q^{(j)} = \log \int p(x^{(j)}) \prod_{(i,t) \in \mathcal{I}_j} \left( (2\pi r^{(j)})^{\frac{1}{2}} (1 - h_{i,t}^{(j)})^{\frac{1}{2}} (h_{i,t}^{(j)})^{\frac{1}{2}} \tilde{p}(y_{i,t} | x_t^{(j)}, z_{i,t} = j) \right) dx^{(j)}.
\]

Since, under the restricted LDS, the likelihood of the observation \( Y_j = [y_{i,t}]_{(i,t) \in \mathcal{I}_j} \) is

\[
\tilde{p}_j(Y_j) = \int p(x^{(j)}) \prod_{(i,t) \in \mathcal{I}_j} \tilde{p}(y_{i,t} | x_t^{(j)}, z_{i,t} = j) dx^{(j)}, \tag{V.144}
\]

it follows that

\[
\log Z_q^{(j)} = \log \left[ \tilde{p}_j(Y_j) \prod_{(i,t) \in \mathcal{I}_j} (2\pi r^{(j)})^{\frac{1}{2}} (1 - h_{i,t}^{(j)})^{\frac{1}{2}} (h_{i,t}^{(j)})^{\frac{1}{2}} \right] \tag{V.145}
= \frac{1}{2} \sum_{(i,t) \in \mathcal{I}_j} (1 - h_{i,t}^{(j)}) \log(2\pi r^{(j)}) - \frac{1}{2} \sum_{(i,t) \in \mathcal{I}_j} \log h_{i,t}^{(j)} + \log \tilde{p}_j(Y_j). \tag{V.146}
\]
Lower bound on $p(Y)$: To lower bound $p(Y)$ as in (V.45), we compute the $\mathcal{L}$ function of (V.32). We start with

$$\log \frac{q(X, Z)}{p(X, Y, Z)} = \log q(X, Z) - \log p(X, Y, Z)$$

(V.147)

$$= \left[ \sum_j \log q(x^{(j)}) + \sum_{i,t} \log q(z_{i,t}) \right]$$

(V.148)

$$- \left[ \sum_{j,i,t} z^{(j)}_{i,t} \log p(y_{i,t} | x^{(j)}_t, z_{i,t} = j) + \sum_j \log p(x^{(j)}) + \log p(Z) \right].$$

Substituting the optimal $q^*$ of (V.61) and (V.62),

$$\log \frac{q(X, Z)}{p(X, Y, Z)} = \left[ \sum_{j,i,t} h^{(j)}_{i,t} \log p(y_{i,t} | x^{(j)}_t, z_{i,t} = j) + \sum_j \log p(x^{(j)}) \right]$$

(V.149)

$$- \sum_j \log \mathcal{Z}^{(j)} + \sum_{j,i,t} z^{(j)}_{i,t} \log h^{(j)}_{i,t} \right] - \left[ \sum_{j,i,t} z^{(j)}_{i,t} \log p(y_{i,t} | x^{(j)}_t, z_{i,t} = j)$$

$$+ \sum_j \log p(x^{(j)}) + \log p(Z) \right].$$

Note that

$$\log p(Z) = \sum_{j,i,t} z^{(j)}_{i,t} \log \alpha^{(j)}_{i,t} + \sum_t \sum_{(i,i') \in \mathcal{E}_t} \left( \log \gamma_2 + \sum_j z^{(j)}_{i,t} z^{(j)}_{i',t} \log \frac{\gamma_1}{\gamma_2} \right)$$

$$+ \sum_i \sum_{(t,t') \in \mathcal{E}_i} \left( \log \beta_2 + \sum_j z^{(j)}_{i,t} z^{(j)}_{i,t'} \log \frac{\beta_1}{\beta_2} \right) - \log \mathcal{Z}. $$

Hence,

$$\log \frac{q(X, Z)}{p(X, Y, Z)} = \sum_{j,i,t} (h^{(j)}_{i,t} - z^{(j)}_{i,t}) \log p(y_{i,t} | x^{(j)}_t, z_{i,t} = j)$$

(V.150)

$$- \sum_j \log \mathcal{Z}^{(j)} + \sum_{j,i,t} z^{(j)}_{i,t} \log \frac{h^{(j)}_{i,t}}{\alpha^{(j)}_{i,t}} - \sum_t \sum_{(i,i') \in \mathcal{E}_t} \left( \log \gamma_2 + \sum_j z^{(j)}_{i,t} z^{(j)}_{i',t} \log \frac{\gamma_1}{\gamma_2} \right)$$

$$- \sum_i \sum_{(t,t') \in \mathcal{E}_i} \left( \log \beta_2 + \sum_j z^{(j)}_{i,t} z^{(j)}_{i,t'} \log \frac{\beta_1}{\beta_2} \right) + \log \mathcal{Z}. $$
Taking the expectation of (V.150) with respect to \( q^*(X, Z) \) yields the KL divergence,

\[
D(q(X, Z) \| p(X, Y, Z)) = - \sum_j \log Z^{(j)}_q + \sum_{j, i, t} h^{(j)}_{i, t} \log \frac{h^{(j)}_{i, t}}{\alpha^{(j)}_{i, t}} \quad (V.151)
\]

\[
- \sum_t \sum_{(i, i') \in \mathcal{E}_t} \left( \log \gamma_2 + \sum_j h^{(j)}_{i, t} h^{(j)}_{i', t} \log \frac{\gamma_1}{\gamma_2} \right)
\]

\[
- \sum_i \sum_{(t, t') \in \mathcal{E}_i} \left( \log \beta_2 + \sum_j h^{(j)}_{i, t} h^{(j)}_{i, t'} \log \frac{\beta_1}{\beta_2} \right) + \log Z_Z.
\]

Substituting into (V.45) yields the log-likelihood lower-bound (V.68).
Chapter VI

Kernel dynamic textures
VI.A Introduction

In the previous chapters, we have shown great success in utilizing the dynamic texture and its multi-modal extensions for video classification, video clustering, and motion segmentation. Despite these numerous successes, one major disadvantage of the dynamic texture is that it can only model video where the motion is smooth, i.e. video textures where the pixel values change smoothly. This limitation stems from the linear assumptions of the model: specifically, 1) the linear state-transition function, which models the evolution of the hidden state-space variables over time; and 2) the linear observation function, which maps the state-space variables into observations. As a result, the dynamic texture is not capable of modeling more complex motion, such as chaotic motion (e.g. turbulent water) or camera motion (e.g. panning, zooming, and rotations).

To some extent, the smoothness limitation of the dynamic texture has been addressed in the literature by modifying the linear assumptions of the dynamic texture model. For example, the model in [98] keeps the linear observation function, while modeling the state transitions with a closed-loop dynamic system that uses a feedback mechanism to correct for errors with a reference signal. In contrast, the models of [99, 100] utilize a non-linear observation function, while keeping the standard linear state transitions. The observation function is modeled as a mixture of linear subspaces (i.e. a mixture of PCA), and the local PCA coordinates are combined into a global coordinate system that represents the state-space. Similarly in [101], different views of a video texture are represented by a non-linear observation function that models the manifold of the video texture from different camera viewpoints. In practice, learning such an observation function is difficult since samples from many different camera views must be taken to learn the function effectively. Finally, [17] treats the observation function as a piece-wise linear function that changes over time. The linear function at time $t$ is computed using a small time-window centered around $t$. The function cannot be computed
outside the given frames of the video, and hence the model is not generative.

In this chapter, we improve the modeling capability of the dynamic texture by using a non-linear observation function, while maintaining the linear state transitions\(^1\). In particular, instead of using PCA to learn a linear observation function, as with the standard dynamic texture, we use *kernel* PCA to learn a non-linear observation function. The resulting *kernel dynamic texture* is capable of modeling a wider range of video motion. The contributions of this chapter are three-fold. First, we introduce the kernel dynamic texture and describe a simple algorithm for learning the parameters of the system. Second, we derive the necessary steps to compute the Martin distance between kernel dynamic textures, and hence introduce a similarity measure for the new model. Third, we build a video classifier based on the kernel dynamic texture and the Martin distance, and evaluate the efficacy of the model through a classification experiment on video containing camera motion. We begin the chapter with a brief review of kernel PCA, followed by each of the three contributions listed above.

### VI.B Kernel PCA

Kernel PCA \([102]\) is the kernelized version of standard PCA \([37]\). With standard PCA, the data is projected onto the linear subspace (linear principal components) that best captures the variability of the data. In contrast, kernel PCA (KPCA) projects the data onto non-linear functions in the input-space. These non-linear principal components are defined by the kernel function, but are never explicitly computed. An alternative interpretation is that kernel PCA first applies a non-linear feature transformation to the data, and then performs standard PCA in the feature-space.

\(^1\)In control theory, these systems are also known as Hammerstein models.
VI.B.1 Learning KPCA coefficients

Given a training data set of \( N \) points \( Y = \{y_1, \ldots, y_N\} \) with \( y_i \in \mathbb{R}^m \) and a kernel function \( k(y_1, y_2) \) with associated feature transformation \( \phi(y) \), i.e. \( k(y_1, y_2) = \langle \phi(y_1), \phi(y_2) \rangle \), the kernel principal components are the eigenvectors of the covariance matrix of the transformed data \( \phi(y_i) \). Assuming that the transformed data is centered (i.e. has zero mean in the feature space), the \( c \)-th principal component in the feature-space has the form [102]

\[
v_c = \sum_{i=1}^{N} \alpha_{i,c} \phi(y_i). \tag{VI.1}
\]

The KPCA weight vector \( \alpha_c = [\alpha_{1,c}, \ldots, \alpha_{N,c}]^T \) is computed as

\[
\alpha_c = \frac{1}{\sqrt{\lambda_c}} v_c, \tag{VI.2}
\]

where \( \lambda_c \) and \( v_c \) are the \( c \)-th largest eigenvalue and corresponding eigenvector of the kernel matrix \( K \), which has entries \( [K]_{i,j} = k(y_i, y_j) \). The scaling of \( v_c \) ensures that the principal component in the feature-space is unit length.

Given the training data point \( y_j \), the \( c \)-th KPCA coefficient \( x_{c,j} \) is computed as the projection of \( \phi(y_j) \) onto the principal component \( v_c \), i.e.

\[
x_{c,j} = \langle \phi(y_j), v_c \rangle = \sum_{i=1}^{N} \alpha_{i,c} k(y_i, y_j), \tag{VI.3}
\]

and hence, the KPCA coefficients \( X = [x_1 \cdots x_N] \) of the training set \( Y \) can be computed as \( X = \alpha^T K \), where \( \alpha = [\alpha_1, \cdots, \alpha_n] \) is the KPCA weight matrix, and \( n \) is the number of principal components.

We have assumed that the transformed data is centered in the feature-space. In the general case, the transformed data must be centered explicitly by using a centered kernel (see Appendix VI.H.1 for more details).

VI.B.2 Reconstruction from KPCA coefficients

KPCA directly models the mapping from the input-space to the KPCA coefficient-space. However since the principal components are never actually computed explicitly (only the projections on to them), the reverse mapping from the
coefficient-space to the input-space is not as straightforward to compute. Given the KPCA coefficients, \( x_t = [x_{1,t}, \ldots, x_{n,t}]^T \), the KPCA reconstruction problem is to find the pre-image \( y_t \) in the input-space that generated these coefficients. In general, this is an ill-posed problem since no \( y_t \) could exist for some KPCA coefficients [103].

The minimum-norm reconstruction method [104, 103] aims to find the pre-image \( y_t \) that minimizes the norm of the error in the feature-space,

\[
y_t^* = \arg\min_{y_t} \| \phi(y_t) - \sum_{c=1}^{n} x_{c,t} v_c \|^2 \quad \text{(VI.4)}
\]

\[
y_t^* = \arg\min_{y_t} k(y_t, y_t) - 2 \sum_{i=1}^{N} \gamma_i k(y_t, y_i), \quad \text{(VI.5)}
\]

where \( \gamma_i = \sum_{c=1}^{n} x_{c,t} \alpha_{i,c} \). When the kernel is the Gaussian kernel, \( k_g(y_1, y_2) = \exp(-\frac{1}{2\sigma^2} \|y_1 - y_2\|^2) \), a solution can be found using an iterative fixed-point procedure [103]. Given some initial guess \( y_t^{(0)} \), refinements are computed using

\[
y_t^{(j+1)} = \frac{\sum_{i=1}^{N} \gamma_i k_g(y_t^{(j)}, y_i) y_i}{\sum_{i=1}^{N} \gamma_i k_g(y_t^{(j)}, y_i)}, \quad \text{(VI.6)}
\]

where \( y_t^{(j)} \) is the estimate of \( y_t^* \) at iteration \( j \). In practice, the initial guess \( y_t^{(0)} \) can be initialized using nearest neighbors in the coefficient space, i.e. choosing \( y_t^{(0)} = y_{i^*} \) such that \( i^* = \arg\min_i \|x_t - x_i\|^2 \). Minimum-norm reconstruction has been shown to be useful in image de-noising applications [103]. For a centered-kernel, the minimum-norm solution can be computed by modifying the \( \gamma_i \) weights (see Appendix VI.H.1 for more details).

Alternatively, reconstruction can also be achieved using other methods, e.g. those based on distance constraints [105], a non-iterative approximation to the fixed-point procedure [106], the Nyström extension [107], or by explicitly modeling the function between the KPCA coefficients and the input-space, e.g. using kernelized ridge regression [108].
VI.C Kernel dynamic textures

In this section, we introduce the kernel dynamic texture. Consider the extension of the standard dynamic texture where the observation matrix $C$ is replaced by a non-linear function $C(x_t)$ of the current state $x_t$,

$$
\begin{align*}
    x_t &= Ax_{t-1} + v_t \\
    y_t &= C(x_t) + w_t.
\end{align*}
$$

(VI.7)

In general, learning the non-linear observation function can be difficult since the state variables are unknown. As an alternative, the inverse of the observation function, i.e. the function $D(y) : \mathbb{R}^m \rightarrow \mathbb{R}^n$ that maps observations to the state-space, can be learned using kernel PCA. The estimates of the state variables are then the KPCA coefficients, and the state-space parameters can be estimated with the least-squares method from Section II.D.2. The details of the learning algorithm are presented in Algorithm 7. We call a non-linear dynamic system, learned in this manner, a kernel dynamic texture (KDT) because it uses kernel PCA to learn the state-space variables, rather than PCA as with the standard dynamic texture. Indeed when the kernel function is the linear kernel, Algorithm 7 reduces to standard least-squares algorithm (Algorithm 2).

The kernel dynamic texture has two interpretations: 1) kernel PCA learns the non-linear observation function $C(x)$, which is composed of a set of non-linear principal components; 2) kernel PCA first transforms the data with the feature-transformation $\phi(y)$ induced by the kernel function, and then a standard dynamic texture is learned in the feature-space. This feature-space interpretation will prove useful in Section VI.D, where we show how to compute the Martin distance between kernel dynamic textures.

VI.C.1 Synthetic examples

In this section we show the expressive power of the kernel dynamic texture on some simple synthetic time-series. Figure VI.1 (left) shows three two-
Algorithm 7 Learning a kernel dynamic texture

**Input:** Video sequence \((y_1, \ldots, y_N)\), state space dimension \(n\), kernel function \(k(y_1, y_2)\).

- Compute the mean: \(\bar{y} = \frac{1}{N} \sum_{i=t}^{N} y_i\).
- Subtract the mean: \(y_t \leftarrow y_t - \bar{y}, \forall t\).
- Compute the (centered) kernel matrix \(K\), where \([K]_{i,j} = k(y_i, y_j)\).
- Compute KPCA weights \(\alpha\) from \(K\).

\[
[x_1 \cdots x_N] = \alpha^T K.
\]

\[
\hat{A} = [\hat{x}_2 \cdots \hat{x}_N][\hat{x}_1 \cdots \hat{x}_{N-1}]^\dagger.
\]

\[
\hat{v}_t = \hat{x}_t - \hat{A}\hat{x}_{t-1}, \forall t.
\]

\[
\hat{Q} = \frac{1}{N-1} \sum_{t=1}^{N-1} \hat{v}_t\hat{v}_t^T.
\]

\[
\hat{y}_t = C(\hat{x}_t), \forall t, \text{ using minimum-norm reconstruction}.
\]

\[
\hat{r} = \frac{1}{mN} \sum_{t=1}^{N} \|y_t - \hat{y}_t\|^2.
\]

**Output:** \(D(y), C(x), \hat{A}, \hat{Q}, \hat{r}, \bar{y}\).

dimensional time-series: 1) a sine wave, 2) a triangle wave, and 3) a periodic ramp wave. Each time-series has length 80, and contains two periods of the waveform. The two-dimensional time-series was projected linearly into a 24-dimensional space, and Gaussian i.i.d. noise with standard deviation \(\sigma = 0.01\) was added. Note that the triangle wave and periodic ramp are not smooth signals in the sense that the former has a discontinuity in the first derivative, while the latter has a jump-discontinuity in the signal.

A dynamic texture and a kernel dynamic texture, using the centered Gaussian kernel with bandwidth determined by (VI.24), were learned from the 24-dimensional time-series, with state-space dimension \(n = 8\). Next, a random sample of length 160 was generated from the dynamic texture and the kernel dynamic texture, and the 24-dimensional signal was linearly projected back into two dimensions for visualization. Figure VI.1 shows the synthesis results for each time-series. The kernel dynamic texture is able to model all three time-series well, including the more difficult triangle and ramp waves. This is in contrast to the
dynamic texture, which can only represent the sine wave. For the triangle wave, the dynamic texture fails to capture the sharp peaks of the triangles, and the signal begins to degrade after $t = 80$. The dynamic texture does not capture any discernible signal from the ramp wave.

The results from these simple experiments indicate that the kernel dynamic texture is much better at modeling arbitrary time-series. In particular, the kernel dynamic texture can model both discontinuities in the first derivative of the signal (e.g. the triangle wave), and jump discontinuities in the signal (e.g. the periodic ramp). These types of discontinuities occur frequently in video textures containing chaotic motion (e.g. turbulent water), or in texture undergoing some camera motion (e.g. panning across a sharp edge). While the application of the kernel dynamic texture to video synthesis is certainly interesting and a direction of future work, in the remainder of this chapter we will focus only on using the model for classification of video textures.
VI.C.2 Tangent distance kernels

One advantage of kernel methods is that a kernel function can be selected that uses a priori information about the underlying data-points (e.g. text classification using string kernels). For kernel dynamic textures, the data-points are images, and hence a kernel function designed for images can be employed. One such kernel is the tangent distance kernel [109], which is based on a tangent distance [110] function that is invariant to image transformations (e.g. translation, scaling, and rotation). When combined with the KDT, the invariance of the tangent kernel allows the KDT to better model video undergoing transformations, e.g. a panning texture video.

Consider an image transformation \( t(y, p) \), parameterized by \( p \in \mathbb{R}^d \), that maps image vectors \( y \in \mathbb{R}^m \) to image vectors \( \mathbb{R}^m \), such \( t(y, 0) = y \). Given an image \( y \), the transformation \( t \) induces a manifold \( \mathcal{M}_y = \{ t(y, p) | p \in \mathbb{R}^d \} \). In the ideal case, the transformation invariant distance between two images, \( y_1 \) and \( y_2 \), is the minimum distance between the induces manifolds,

\[
d^2_{\mathcal{M}}(y_1, y_2) = \min_{p_1, p_2} \| t(y_1, p_1) - t(y_2, p_2) \|^2.
\]  

To reduce computational complexity, we can approximate the manifold \( \mathcal{M}_y \) by its tangent hyperplane \( T \) at point \( y \)

\[
\mathcal{M}_y \approx \{ y + Tp, | p \in \mathbb{R}^d \},
\]

where \( t_i = \frac{\partial t(x, p_i)}{\partial p_i} \bigg|_{p=0} \) are the tangents vectors with respect to each parameter \( p_i \), and \( T = [t_1, \cdots, t_d] \in \mathbb{R}^{m \times d} \). Substituting (VI.9) into (VI.8), yields the “two-sided” tangent distance [111, 110]

\[
d^2_T(y_1, y_2) = \min_{p_1, p_2} \| y_1 + T_1p_1 - y_2 - T_2p_2 \|^2,
\]

where \( T_1 \) and \( T_2 \) are the tangent vectors for \( y_1 \) and \( y_2 \). (VI.10) computes the minimum distance between the two tangent hyperplanes. This is a linear least-
squares problem, with solution obtained by solving the system of equations

\[(T_{21}T_{11}^{-1}T_1^T - T_2^T)(y_1 - y_2) = (T_{21}T_{11}^{-1}T_{12} - T_{22})p_2,\]  
\[(T_{12}T_{22}^{-1}T_2^T - T_1^T)(y_1 - y_2) = (T_{11} - T_{12}T_{22}^{-1}T_{21})p_1,\]  

(VI.11)

where \(T_{ij} = T_i^T T_j\) for \(i, j = \{1, 2\}\). In practice, the system of equations might be singular if the two tangent hyperplanes are parallel, and hence a penalty term is added to (VI.10) to prevent a degenerate solution

\[d_T^2(y_1, y_2) = \min_{p_1, p_2} \|y_1 + T_1p_1 - y_2 - T_2p_2\|^2 + k(\|T_1p_1\|^2 + \|T_2p_2\|^2),\]  

(VI.12)

where \(k\) is a regularization parameter. The solution to (VI.12) is obtained by solving

\[(T_{21}T_{11}^{-1}T_1^T - (1 + k)T_2^T)(y_1 - y_2) = (T_{21}T_{11}^{-1}T_{12} - (1 + k)^2T_{22})p_2,\]  
\[(T_{12}T_{22}^{-1}T_2^T - (1 + k)T_1^T)(y_1 - y_2) = ((1 + k)^2T_{11} - T_{12}T_{22}^{-1}T_{21})p_1,\]  

(VI.13)

The tangent distance has primarily been applied to USPS digit recognition [110, 112, 113], and object and speech recognition [114].

Finally, tangent distance kernels [109] are obtained by replacing the Euclidean distance with the tangent distance in any of the standard distance-based kernels. Of particular interest is the modification of the RBF kernel,

\[k_T(y_1, y_2) = e^{-\gamma d_T^2(y_1, y_2)}.\]  

(VI.14)

Note that, since the tangent distance is not a metric (it does not satisfy the triangle inequality), the above kernel is not positive definite. Nonetheless, such kernels are often applicable and have been successfully applied to a wide range of problems, e.g. the KL kernel [38], dynamic time warping kernel [115], or sigmoid-kernel [116].

**VI.C.3 Related works**

The kernel dynamic texture is related to non-linear dynamical systems, where both the state transitions and the observation functions are non-linear functions. In [117], the EM algorithm and the extended Kalman filter are used to learn
the parameters of a non-linear dynamical system. In [118], the nonlinear mappings are modeled as multi-layer perceptron networks, and the system is learned using a Bayesian ensemble method. These methods can be computationally intensive because of the many degrees of freedom associated with both non-linear state transitions and non-linear observation functions. In contrast, the kernel dynamic texture is a model where the state transition is linear and the observation function is non-linear.

The kernel dynamic texture also has connections to dimensionality reduction; several manifold-embedding algorithms (e.g. ISOMAP, LLE) can be cast as kernel PCA with kernels specific to each algorithm [119]. Finally, the kernel dynamic texture is similar to [120, 121], which learns appearance manifolds of video that are constrained by a Gauss-Markov state-space with known parameters $A$ and $Q$.

### VI.D The Martin distance between kernel dynamic textures

In this section, we extend the Martin distance to kernel dynamic textures, enabling classification and comparison of kernel dynamic textures. Let $\Theta_a = \{C_a, A_a\}$ and $\Theta_b = \{C_b, A_b\}$ be the parameters of two dynamic textures. Recall from Section III.D, the Martin distance is defined as

$$d^2(\Theta_a, \Theta_b) = -2 \sum_{i=1}^{n} \log \lambda_i, \quad (VI.15)$$

where $\lambda_i$ are the first $n$ largest eigenvalues of the generalized eigenvalue problem:

$$
\begin{bmatrix}
0 & \mathcal{O}_{ab} \\
(\mathcal{O}_{ab})^T & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix} = \lambda
\begin{bmatrix}
\mathcal{O}_{aa} & 0 \\
0 & \mathcal{O}_{bb}
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix} \quad (VI.16)
$$

subject to $x^T \mathcal{O}_{aa} x = 1$ and $y^T \mathcal{O}_{bb} y = 1$, where

$$
\mathcal{O}_{ab} = (\mathcal{O}_a)^T \mathcal{O}_b = \sum_{t=0}^{\infty} (A_t^a)^T C_a T C_b A_t^b, \quad (VI.17)
$$
and similarly for $O_{aa}$ and $O_{bb}$. The Martin distance for kernel dynamic textures can be computed by using the interpretation that the kernel dynamic texture is a standard dynamic texture learned in the feature-space of the kernel. Hence, in the matrix $F = C^T C_b$, the inner-products between the principal components can be replaced with the inner-products between the kernel principal components in the feature-space. However, this can only be done when the two kernels induce the same inner-product in the same feature-space.

Consider two data sets $\{y_{i}^{(a)}\}_{i=1}^{N_a}$ and $\{y_{i}^{(b)}\}_{i=1}^{N_b}$, and two kernel functions $k_a$ and $k_b$ with feature transformations $\phi(y)$ and $\psi(y)$, i.e.

$$k_a(y_1, y_2) = \langle \phi(y_1), \phi(y_2) \rangle, \quad k_b(y_1, y_2) = \langle \psi(y_1), \psi(y_2) \rangle.$$  \hspace{1cm} (VI.18)

Running KPCA on each of the data-sets with their kernels yields the KPCA weight matrices $\alpha$ and $\beta$, respectively. The c-th and d-th KPCA components in each of the feature-spaces is given by,

$$u_c = \sum_{i=1}^{N_a} \alpha_{i,c} \phi(y_{i}^{(a)}), \quad v_d = \sum_{i=1}^{N_b} \beta_{i,d} \psi(y_{i}^{(b)}).$$  \hspace{1cm} (VI.19)

Hence, the inner-product between these two KPCA components is

$$\langle u_c, v_d \rangle = \left\langle \sum_{i=1}^{N_a} \alpha_{i,c} \phi(y_{i}^{(a)}), \sum_{i=1}^{N_b} \beta_{i,d} \psi(y_{i}^{(b)}) \right\rangle$$  \hspace{1cm} (VI.20)

$$= \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} \alpha_{i,c} \beta_{j,d} g(y_{i}^{(a)}, y_{j}^{(b)})$$  \hspace{1cm} (VI.21)

$$= \alpha^T_c G \beta_d,$$  \hspace{1cm} (VI.22)

where $g(y_1, y_2) = \langle \phi(y_1), \psi(y_2) \rangle$, and $G$ is the corresponding matrix with entries $[G]_{i,j} = g(y_{i}^{(a)}, y_{j}^{(b)})$. The function $g$ can be computed if the two kernels have “compatible” feature-spaces, i.e. the two kernels induce the same inner-product in the feature-space. For example, when each kernel is a Gaussian kernel with bandwidth parameters $\sigma_a^2$ and $\sigma_b^2$, it can be shown that $g(y_1, y_2) = \exp(-\frac{1}{2} \frac{1}{\sigma_a^2} y_1 - \frac{1}{\sigma_b^2} y_2 \|^2)$. The proof appears in the Appendix VI.H. Finally, the inner product
matrix between all the KPCA components is

\[ F = \alpha^T G \beta. \]  \hfill (VI.23)

For a centered kernel, see Appendix VI.H for details.

VI.E \ Applications to video classification

In this section we evaluate the efficacy of the kernel dynamic texture for classification of video textures undergoing camera motion.

VI.E.1 Panning video texture databases

The UCLA dynamic texture database [16] used in Chapter III contains 50 classes of various video textures, including boiling water, fountains, fire, waterfalls, and plants and flowers swaying in the wind. Each class contains four grayscale sequences with 75 frames of 160 × 110 pixels. Each sequence was clipped to a 48 × 48 window that contained the representative motion.

Three additional databases, containing video textures with different levels of panning, were built from the original UCLA video textures. Video textures were generated by panning a 40 × 40 window across the original UCLA video. Four pans (two left pans and two right pans) were generated for each video sequence, resulting 800 panning textures in each database. Each database was generated

![Figure VI.2 Examples from the UCLA-pan-1 video texture database.](image-url)
with a different amount of panning, 0.5 pixels per frame, 1 pixel per frame, and 2 pixels per frame. We denote these databases UCLA-pan-0.5, UCLA-pan-1, and UCLA-pan-2. The motion in these database is composed of both video textures and camera panning, hence the dynamic texture is not expected to perform well on it. Examples of the UCLA-pan-1 database are shown in Figure VI.2, and a montage video of examples from both databases is available from the supplemental [52].

**VI.E.2 Experimental setup**

A kernel dynamic texture was learned for each video in the database using Algorithm 7. Two kernels were used, the centered RBF (Gaussian) kernel and the centered tangent distance kernel in (VI.14) (using tangent vectors for image translation), which we denote KDT-rbf and KDT-tan. In both cases, the bandwidth parameter $\sigma^2$ was computed using the following scale estimate for each video,

$$
\sigma^2 = \frac{1}{2} \text{median}\left\{\|y_i - y_j\|^2\right\}_{i,j=1,...,N} \quad \text{(VI.24)}
$$

Both nearest neighbor (NN) and SVM classifiers [122] were trained using the Martin distance for the kernel dynamic texture. The SVM used an RBF-kernel based on the Martin distance,

$$
k_{md}(\Theta_a, \Theta_b) = e^{-\frac{1}{2\sigma^2}d^2(\Theta_1, \Theta_2)} \quad \text{(VI.25)}
$$

A one-versus-all scheme was used to learn the multi-class SVM problem, and the $C$ and $\gamma$ parameters were selected using three-fold cross-validation over the training set. We used the libsvm package [54] to train and test the SVM.

For comparison, a NN using the Martin distance on the standard dynamic texture [16] was trained, along with a corresponding SVM classifier. A NN and SVM classifier using the KL-divergence between dynamic textures in the image space (Chapter III) were also trained. Finally, experimental results were averaged over four trials, where in each trial the databases was split differently with 75% of data used for training and cross-validation, and 25% of the data used for testing.
VI.E.3 Classification results

Figure VI.3 (left) shows the NN classifier performance versus $n$, the number of principal components (or the dimension of the state space), for each of the four databases. While the NN classifier based on the kernel dynamic texture with RBF kernel and Martin distance (KDT-rbf-MD) performs similarly to the dynamic texture (DT-MD) on the UCLA database, KDT-rbf-MD outperforms DT-MD for all values of $n$ on all three databases with panning. The best accuracy increases 5.5%, 5.3%, and 4.9% on the three panning databases when using KDT-rbf-MD instead of DT-MD, while only increasing from 1.0% on the UCLA database. In addition, the performance of DT-MD decreases monotonically, from 89.0% to 83.6%, as the amount of panning increases. This indicates that the panning motions in these databases are not well modeled by the dynamic texture, whereas the kernel dynamic texture has better success. The performance of the SVM classifiers are shown in Figure VI.3 (right). The dynamic texture and kernel dynamic texture perform similarly on the UCLA database, with both improving over their corresponding NN classifier. However, the KDT-rbf-MD SVM outperforms the DT-MD SVM on the UCLA-pan database (accuracies of 94.1% and 92.8%, respectively). Again, this indicates that the UCLA-pan database is not well modeled by the standard dynamic texture.

Comparing KDT using the two kernels, the tangent distance kernel (KDT-tan-MD) performs worse than the RBF kernel (KDT-rbf-MD) when there is no panning (Figure VI.3a), with accuracies of 90% versus 87% for NN. This can be seen as an over-fitting problem, since the tangent distance kernel is expecting panning when there is none. On the other hand, as the amount of panning increases, KDT-tan-MD improves, and eventually outperforms KDT-rbf-MD (e.g. 91.6% accuracy versus 88.5% on UCLA-pan-2). This suggests that the KDT with tangent distance kernel is better capable of modeling the panning textures, since the kernel is invariant to translations. On the other hand, there is only a modest gain in performance when using the tangent distance kernel because the texture motion
Figure VI.3 Classification results for NN (left) and SVM (right) on panning video texture databases: a) no panning, b) 0.5 pixels per frame, c) 1 pixel per frame, d) 2 pixels per frame. Classifier accuracy is plotted versus the number of principal components $n$. 
may significantly alter the appearance of the next frame, making the recovery of the correct translation parameters difficult.

Table VI.1 Classification results on panning video texture databases. \((n)\) is the number of principal components.

<table>
<thead>
<tr>
<th>Database</th>
<th>KDT-rbf-MD</th>
<th>KDT-tan-MD</th>
<th>DT-MD</th>
<th>DT-KL0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN UCLA</td>
<td>0.900 (20)</td>
<td>0.870 (10)</td>
<td>0.890 (15)</td>
<td>0.370 (2)</td>
</tr>
<tr>
<td>NN UCLA-pan-0.5</td>
<td>0.939 (30)</td>
<td>0.930 (30)</td>
<td>0.884 (25)</td>
<td>0.775 (5)</td>
</tr>
<tr>
<td>NN UCLA-pan-1</td>
<td>0.896 (30)</td>
<td>0.898 (25)</td>
<td>0.843 (30)</td>
<td>0.816 (5)</td>
</tr>
<tr>
<td>NN UCLA-pan-2</td>
<td>0.885 (20)</td>
<td>0.916 (25)</td>
<td>0.836 (20)</td>
<td>0.834 (2)</td>
</tr>
<tr>
<td>SVM UCLA</td>
<td>0.970 (20)</td>
<td>0.965 (20)</td>
<td>0.960 (15)</td>
<td>0.725 (2)</td>
</tr>
<tr>
<td>SVM UCLA-pan-0.5</td>
<td>0.958 (30)</td>
<td>0.953 (30)</td>
<td>0.931 (15)</td>
<td>0.898 (5)</td>
</tr>
<tr>
<td>SVM UCLA-pan-1</td>
<td>0.941 (30)</td>
<td>0.955 (30)</td>
<td>0.928 (25)</td>
<td>0.920 (5)</td>
</tr>
<tr>
<td>SVM UCLA-pan-2</td>
<td>0.950 (20)</td>
<td>0.956 (30)</td>
<td>0.919 (25)</td>
<td>0.928 (10)</td>
</tr>
</tbody>
</table>

When looking at the performance of the KL-based classifiers, we note that for the UCLA databases the mean-image of the video is highly discriminative for classification. This can be seen in Figure VI.3 (a), where the accuracy of the KL-divergence NN classifier is plotted for dynamic textures learned from the normal data (DT-KL), and dynamic textures learned from zero-mean data (DT-KL0). The performance is very good when the mean is included. In fact, the best performance occurs when \(n = 0\), i.e. the video is simply modeled as the mean image with some i.i.d. Gaussian noise. On the other hand, when the mean is ignored, the performance of the classifier drops dramatically (from 96% to 15% accuracy for \(n = 0\)). Hence, much of the discriminative power of the KL-based classifier comes from the similarity of image means, not from video motion. Because the Martin distance does not use the image means, we present the classification results for DT-KL0 to facilitate a fairer comparison between the classifiers. The DT-KL0 NN classifier performed worse than both KDT and DT-MD, as seen in Figure VI.3 (left). The SVM trained on DT-KL0 improved the performance over the DT-KL0 NN classifier, but is still inferior to the KDT-rbf-MD and KDT-tan-MD SVM classifiers. A summary of the results on the four databases is given in Table VI.1.

Finally, Figure VI.4 shows the distance matrix for DT-MD and KDT-
rbf-MD for three classes from UCLA-pan-1. The DT-MD performs poorly on many of these sequences because the water motion is chaotic, i.e. there are many discontinuities in the pixel values. On the other hand, KDT-rbf-MD is able to model the discontinuities, and hence can distinguish between the different types of chaotic water motion.

Figure VI.4 Misclassification of chaotic water: the Martin distance matrices for three water classes using (left) dynamic textures, and (right) kernel dynamic textures. Nearest neighbors in each row are indicated by a black dot, and the misclassifications are circled. (bottom) Four examples from each of the three water classes.

VI.F Summary and discussion

In this chapter, we have addressed the smoothness limitations of the dynamic texture by improving the modeling capability of the dynamic texture,
specifically, by adopting a non-linear observation function in the dynamic texture model. The non-linear function is represented with kernel PCA, and the resulting kernel dynamic texture is capable of modeling a wider range of video motion, such as chaotic motion (e.g. turbulent water) or camera motion (e.g. panning). We derived a distance function between kernel dynamic textures, and showed that the new model improves classification of textures undergoing panning. When a large amount of panning is present, the KDT learned with the tangent distance kernel outperforms the KDT learned with the vanilla Gaussian kernel. However, this gain is only modest because the motion of the texture makes the recovery of the correct translation parameters difficult.

Finally, we note that, while video synthesis from the KDT is certainly an interesting topic for future work, the quality of the video generated from the model is ultimately dependent on the reconstruction algorithm used to solve the kernel pre-image problem. The solutions given by the current methods [104, 103, 105, 106, 107, 108] are essentially convex combinations of the training points, and hence, reconstruction of an arbitrary point on a non-linear manifold requires a dense sampling of the manifold (specifically, a sampling that is dense enough to form a good piece-wise linear approximation of the manifold). For video, where the data is high-dimensional and the underlying manifold is complicated, this dense sampling may be difficult to obtain, or too large to work with.

VI.G Acknowledgements

We thank Benjamin Recht for helpful discussions, and Gianfranco Doretto and Stefano Soatto for the database from [16].

The text of Chapter VI, in part, is based on the material as it appears in: A. B. Chan and N. Vasconcelos, “Classifying video with kernel dynamic textures”, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2007. The dissertation author was a primary researcher and an author of the cited
VI.H Appendix

VI.H.1 Kernel centering

In this section, we derive the “centered kernel” versions of KPCA, minimum norm reconstruction, and the inner-product between KPCA components.

Centering for KPCA

In Section VI.B, we have assumed that the transformed data is centered in the feature-space. In the general case, the transformed data must be centered explicitly by subtracting the empirical mean, resulting in the centered feature transformation

\[
\tilde{\phi}(y) = \phi(y) - \frac{1}{N} \sum_{n=1}^{N} \phi(y_n),
\]

where \(y_j\) are the training data. The centered kernel matrix between two points \(y\) and \(y'\) is then

\[
\tilde{k}(y, y') = \left\langle \tilde{\phi}(y), \tilde{\phi}(y') \right\rangle = \left\langle \phi(y) - \frac{1}{N} \sum_{n=1}^{N} \phi(y_n), \phi(y') - \frac{1}{N} \sum_{n=1}^{N} \phi(y_n) \right\rangle
\]

\[
= k(y, y') - \frac{1}{N} \sum_{n=1}^{N} k(y', y_n) - \frac{1}{N} \sum_{n=1}^{N} k(y_n, y)
\]

\[
+ \frac{1}{N^2} \sum_{n,m} k(y_n, y_m).
\]

Hence, for the training kernel, the centering is obtained from the non-centered kernel [108] as

\[
\tilde{K} = K - \frac{1}{N} 11^T K - \frac{1}{N} K 11^T + \frac{1}{N^2} 11^T K 11^T
\]

\[
= (I - \frac{1}{N} 11^T) K (I - \frac{1}{N} 11^T),
\]
where $\mathbf{1}$ is the vector of $N$ ones. Given a test point $y_t$, the centered kernel between the test point and the training points is obtained from the non-centered kernel $K_t = [k(y_t, y_1), \ldots, k(y_t, y_N)]$ as

$$\tilde{K}_t = K_t - \frac{1}{N} \mathbf{1}^T K - \frac{1}{N} K_t \mathbf{1} \mathbf{1}^T + \frac{1}{N^2} \mathbf{1}^T K \mathbf{1} \mathbf{1}^T \quad \text{(VI.32)}$$

$$= K_t (I - \frac{1}{N} \mathbf{1} \mathbf{1}^T) - \frac{1}{N} \mathbf{1}^T K (I - \frac{1}{N} \mathbf{1} \mathbf{1}^T) \quad \text{(VI.33)}$$

$$= (K_t - \frac{1}{N} \mathbf{1} \mathbf{1}^T K) (I - \frac{1}{N} \mathbf{1} \mathbf{1}^T). \quad \text{(VI.34)}$$

**Centering for minimum-norm reconstruction**

Minimum-norm reconstruction using the centered kernel is given by

$$y^*_t = \arg \min_{y_t} \left\| \tilde{\phi}(y_t) - \sum_c x_{c,t} v_c \right\|^2 \quad \text{(VI.35)}$$

$$= \arg \min_{y_t} \tilde{k}(y_t, y_t) - 2 \sum_c x_{c,t} \left( v_c, \tilde{\phi}(y_t) \right) + \sum_{c,c'} x_{c,t} x_{c',t} \left( v_c, v_{c'} \right) \quad \text{(VI.36)}$$

$$= \arg \min_{y_t} \tilde{k}(y_t, y_t) - 2 \sum_c x_{c,t} \sum_n \alpha_{n,c} \tilde{k}(y_t, y_n) + \sum_c (x_{c,t})^2 \quad \text{(VI.37)}$$

$$= \arg \min_{y_t} \tilde{k}(y_t, y_t) - 2 \sum_n \gamma_n \tilde{k}(y_t, y_n) \quad \text{(VI.38)}$$

$$= \arg \min_{y_t} \left[ k(y_t, y_t) - \frac{2}{N} \sum_n k(y_t, y_n) + \frac{1}{N^2} \sum_{n,m} k(y_n, y_m) \right] \quad \text{(VI.39)}$$

$$- 2 \sum_n \gamma_n \left[ k(y_t, y_n) - \frac{1}{N} \sum_j k(y_j, y_n) - \frac{1}{N} \sum_j k(y_j, y_t) \right. $$

$$\left. + \frac{1}{N^2} \sum_{i,j} k(y_i, y_j) \right]$$

$$= \arg \min_{y_t} \left[ k(y_t, y_t) - \frac{2}{N} \sum_n k(y_t, y_n) - 2 \sum_n \gamma_n k(y_t, y_n) \right. $$

$$\left. + 2 \sum_n \gamma_n \frac{1}{N} \sum_j k(y_t, y_j) \right]$$

$$= \arg \min_{y_t} \left[ k(y_t, y_t) - 2 \sum_n \left( \gamma_n - \frac{1}{N} \sum_i \gamma_i \right) k(y_t, y_n) \right. $$

$$\left. + 2 \sum_n \gamma_n \frac{1}{N} \sum_j k(y_t, y_j) \right]$$

$$= \arg \min_{y_t} \left[ k(y_t, y_t) - 2 \sum_n \left( \gamma_n - \frac{1}{N} \sum_i \gamma_i \right) k(y_t, y_n) \right. $$

$$\left. + 2 \sum_n \gamma_n \frac{1}{N} \sum_j k(y_t, y_j) \right]$$

$$= \arg \min_{y_t} k(y_t, y_t) - 2 \sum_n \left( \gamma_n - \frac{1}{N} \sum_i \gamma_i + \frac{1}{N} \right) k(y_t, y_n). \quad \text{(VI.41)}$$
Hence, the reconstruction problem using the centered kernel reduces to the standard reconstruction problem using the un-centered kernel with modified weights,

\[ \tilde{\gamma}_n = \gamma_n - \frac{1}{N} \sum_i \gamma_i + \frac{1}{N}. \]  
(VI.42)

Centering for the inner-product between KPCA components

Consider two data sets \( \{y^{(a)}_i\}_{i=1}^{N_a} \) and \( \{y^{(b)}_i\}_{i=1}^{N_b} \), and two centered kernel functions \( \tilde{k}_a \) and \( \tilde{k}_b \) with centered feature-transformations \( \tilde{\phi}(y) \) and \( \tilde{\psi}(y) \), i.e.

\[ \tilde{k}_a(y_1, y_2) = \left\langle \tilde{\phi}(y_1), \tilde{\phi}(y_2) \right\rangle, \quad \tilde{k}_b(y_1, y_2) = \left\langle \tilde{\psi}(y_1), \tilde{\psi}(y_2) \right\rangle, \]  
(VI.43)

which share the same inner-product and feature-spaces. Running KPCA on each of the data-sets with their centered kernels yields the KPCA weight matrices \( \alpha \) and \( \beta \), respectively. The \( c \)-th and \( d \)-th KPCA components in each of the feature-spaces are given by,

\[ u_c = \sum_i \alpha_{i,c} \tilde{\phi}(y^{(a)}_i), \quad u_d = \sum_i \beta_{i,d} \tilde{\psi}(y^{(b)}_i). \]  
(VI.44)

Hence, their inner product is given by

\[
\begin{align*}
\langle u_c, u_d \rangle &= \left\langle \sum_i \alpha_{i,c} \tilde{\phi}(y^{(a)}_i), \sum_j \beta_{j,d} \tilde{\psi}(y^{(b)}_j) \right\rangle \\
&= \sum_{i,j} \alpha_{i,c} \beta_{j,d} \left\langle \tilde{\phi}(y^{(a)}_i), \tilde{\psi}(y^{(b)}_j) \right\rangle \\
&= \sum_{i,j} \alpha_{i,c} \beta_{j,d} \left( g(y^{(a)}_i, y^{(b)}_j) - \frac{1}{N_a} \sum_k g(y^{(a)}_k, y^{(b)}_i) - \frac{1}{N_b} \sum_k g(y^{(a)}_i, y^{(b)}_k) \right) \\
&= \alpha^T_c G \beta_d - \left( \frac{1^T \alpha_c}{N_a} \right) \frac{1^T G \beta_d}{N_b} - \left( \frac{1^T \beta_d}{N_b} \right) \alpha^T_c G 1 \\
&\quad + \left( \frac{1^T \alpha_c}{N_a} \right) \left( \frac{1^T \beta_d}{N_b} \right) N_a N_b 1^T G 1 \\
&= \left( \alpha_c - \frac{1^T \alpha_c}{N_a} \right) G \left( \beta_d - \frac{1^T \beta_d}{N_b} \right) 1,
\end{align*}
\]  
(VI.49)
where \( g(y_1, y_2) = \langle \phi(y_1), \psi(y_2) \rangle \), and the matrix \( G \) has entries \([G]_{i,j} = g(y_i^{(a)}, y_j^{(b)})\). Hence, the computation for the centered kernel is equivalent to using the non-centered kernel, but with \( \alpha_c \) and \( \beta_d \) normalized by subtracting their respective means

\[
\tilde{\alpha}_c = \alpha_c - \frac{(1^T \alpha_c)}{N_a} \mathbf{1}, \quad \tilde{\beta}_d = \beta_d - \frac{(1^T \beta_d)}{N_b} \mathbf{1}. \tag{VI.51}
\]

**Summary**

Table VI.2 lists the modifications when using a centered kernel with the various KPCA algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Modification for centering</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPCA training kernel</td>
<td>( K = (I - \frac{1}{N} \mathbf{1} \mathbf{1}^T)K(I - \frac{1}{N} \mathbf{1} \mathbf{1}^T) )</td>
</tr>
<tr>
<td>KPCA testing kernel</td>
<td>( \tilde{K}_t = (K_t - \frac{1}{N} \mathbf{1} \mathbf{1}^T K)(I - \frac{1}{N} \mathbf{1} \mathbf{1}^T) )</td>
</tr>
<tr>
<td>KPCA min-norm reconstruction</td>
<td>( \tilde{\gamma}<em>i = \gamma_i - \frac{1}{N} \sum</em>{j=1}^N \gamma_j + \frac{1}{N} )</td>
</tr>
<tr>
<td>inner-product btwn. KPCA comp.</td>
<td>( \tilde{\alpha}_c = \alpha_c - \frac{(1^T \alpha_c)}{N_a} \mathbf{1}, \tilde{\beta}_d = \beta_d - \frac{(1^T \beta_d)}{N_b} \mathbf{1} )</td>
</tr>
</tbody>
</table>

**VI.H.2 Inner-product between Gaussian feature-spaces**

In this section, we derive the inner-product between the feature transformations of two Gaussian kernels with different bandwidth parameters. Let \( \Phi(y) \) be the feature transformation induced by the Gaussian kernel with unit variance, i.e.

\[
k(y_1, y_2) = e^{-\frac{1}{2} \| y_1 - y_2 \|^2} = \langle \Phi(y_1), \Phi(y_2) \rangle . \tag{VI.52}
\]

Now consider the Gaussian kernel parameterized by \( \sigma^2 \),

\[
k_{\sigma}(y_1, y_2) = e^{-\frac{1}{2 \sigma^2} \| y_1 - y_2 \|^2} = e^{-\frac{1}{2} \| \frac{1}{\sigma} y_1 - \frac{1}{\sigma} y_2 \|^2} \tag{VI.53}
\]

\[
= \left\langle \Phi \left( \frac{1}{\sigma} y_1 \right), \Phi \left( \frac{1}{\sigma} y_2 \right) \right\rangle . \tag{VI.54}
\]

Hence, the feature transformation of the Gaussian with variance \( \sigma^2 \) is related to the Gaussian kernel with unit variance via \( \Phi_{\sigma}(y) = \Phi \left( \frac{1}{\sigma} y \right) \). Finally, for two
Gaussian kernels parameterized by $\sigma_a^2$ and $\sigma_b^2$, the inner product between their feature-transformations is

$$g(y_1, y_2) = \langle \Phi_a(y_1), \Phi_b(y_2) \rangle$$  \hspace{1cm} (VI.55)

$$= \left\langle \Phi \left( \frac{1}{\sigma_a} y_1 \right), \Phi \left( \frac{1}{\sigma_b} y_2 \right) \right\rangle$$  \hspace{1cm} (VI.56)

$$= e^{-\frac{1}{2} \| \frac{1}{\sigma_a} y_1 - \frac{1}{\sigma_b} y_2 \|^2}.$$  \hspace{1cm} (VI.57)
Chapter VII

Application: Background subtraction for dynamic scenes
VII.A Introduction

Background subtraction is an important first step for many vision problems. It separates objects from background clutter, usually by comparing motion patterns, and facilitates subsequent higher-level operations, such as tracking, object identification, etc. Because the environment can change substantially, both in the short term and throughout the lifetime of the vision system, background subtraction algorithms are expected to be robust. This is not always easy to guarantee, and many methods have been proposed in the literature (see [123] for a recent review). One approach, which was first introduced by Stauffer and Grimson (SG) in [124] and has since gained substantial popularity, is to model the distribution of colors (over time) of each pixel as a mixture of Gaussians. This accounts for the fact that, in scenes with multiple objects, pixel colors change as objects traverse the scene. For example, if an object stops, it should at some point be considered part of the background. As it departs, the un-occluded area should be quickly reassigned to the background. Some objects may even exhibit cyclic motion, e.g. a flickering light display, making a number of background pixels undergo cyclic variations of color over time. The mixture model captures these state transitions very naturally, while providing a compact summary of the color distribution. This simplifies the management of the background model, namely the problems of updating the distribution over time, or deciding which components of the mixture model should be dropped as the background changes (due to variable scene lighting, atmospheric conditions, etc.).

Nevertheless, the algorithm proposed by SG is not without problems. One of its main drawbacks is the assumption that the background is static over short time scales. This is a strong limitation for scenes with spatio-temporal dynamics, such as those of Figure VII.1. Although the model allows each pixel to switch state, and tolerates some variability within the state, the Gaussian mixture assumes that the variability derives from noise, not the structured motion patterns that
characterize moving water, burning fire, swaying trees, etc. One approach that has shown promise for modeling these spatio-temporal dynamic processes is the \textit{dynamic texture} representation. Not surprisingly, background models based on dynamic textures, or close relatives, have appeared in the literature. These utilize dynamic textures to model the whole video \cite{125}, or video patches extracted from the neighborhood of each location \cite{126}.

Figure VII.1 A scene with dynamic background. The background consists of water waves, which are changed by the turbulent wake of a boat.

While able to capture background dynamics, these approaches lack the two most compelling (and dynamic) aspects of the SG method: 1) the ability to account for transitory events, due to motion of foreground objects; and 2) simple model management. Consider, for example, the aquatic scene of Figure VII.1. As the jet-skier traverses a video patch, the video goes through the following state sequence: normal waves, occluded by jet-ski, turbulent waves that trail the jet-ski, return to normal waves. In the absence of a hidden discrete state variable, the dynamic texture will slowly interpolate through all these states. Both the transition from occluded to turbulent, and turbulent to normal waves, will generate outliers which are incorrectly marked as foreground. If the jet-ski cyclically passes through the same location, these errors will repeat with the same periodicity.

In summary, background subtraction requires both a \textit{state-based} representation (as in SG) and the ability to \textit{capture scene dynamics} within the state (as in the dynamic texture methods). This suggests a very natural extension of the two lines of work: to represent spatio-temporal video cubes as samples from a \textit{mixture of dynamic textures}. However, as in the static case, exact learning of the mixture parameters is computationally infeasible in an on-line setting. To address
this problem, we combine two observations. The first is the main insight of SG: that parameter updates of a Gaussian mixture model only require a small set of \textit{sufficient statistics}. The second is that, because the dynamic texture is a member of the exponential family [24], it exhibits the same property. Hence, we use the sufficient statistics and the on-line learning algorithm, derived in Chapter II, to \textit{generalize the SG algorithm} to dynamic scenes. The generalized SG (GSG) algorithm inherits the advantages of [124]: 1) it adapts to long-term variations via on-line estimation; 2) it can quickly embrace new background motions through the addition of mixture components; and 3) it easily discards outdated information by dropping mixture components with small priors. Experimental results show that background modeling with the adaptive mixture of dynamic textures substantially outperforms both static background models, and those based on a single dynamic texture.

The remainder of this chapter is organized as follows. Section VII.B briefly reviews related work in background subtraction. In Section VII.C, we review the SG method and introduce the proposed generalization. In Section VII.D, we introduce the adaptive background subtraction algorithm based on mixtures of dynamic textures. Finally, an experimental evaluation is described in Section VII.E.

VII.B Related work

A number of extensions to the background subtraction method of SG have appeared in the literature. One of the main drawbacks of the original approach was the difficulty of detecting foreground objects of color similar to that of the background. This motivated a number of extensions based on properties of local image neighborhoods, e.g. texture information [127], optical flow [128, 129], or spatio-temporal blocks [130, 131]. Another drawback of [124] is the lack of consistency between the state of adjacent pixels, which sometimes leads to noisy
foreground-background segmentations. This motivated various extensions that encourage global consistency, either through global image modeling (e.g. eigenbackground methods [132, 133]), by introducing priors that enforce global consistency (e.g. the combination of Markov random fields and maximum a posteriori probability decision rules [123]), or by allowing pixels to influence neighboring GMMs [134]. While most of these models can adapt to slow background changes (e.g. due to illumination) they share, with the mixture model of SG, the assumption that the background is *static* over short time scales.

In the realm of *dynamic models*, two main methods have been proposed. In [125], the video is modeled with a robust Kalman filter. A dynamic texture models the entire video frame, and pixels that are not well explained by the LDS (i.e. outliers), conditioned on the previous frame, are marked as foreground. This method has limited effectiveness, because a global fit usually demands excessive representational power from the dynamic texture: background dynamics are seldom homogeneous throughout the scene, due to a combination of 3-D effects (perspective and motion parallax) and background variability (e.g. trees swaying with different dynamics, or waves with normal and turbulent motions). A natural improvement is to rely on a localized representation. This was first proposed in [126], where the dynamic texture is applied to video patches, reducing the dimensionality of the model and producing a simpler learning problem. This work also exploits the principal component analysis (PCA) performed within the DT to achieve computational efficiency. In particular, all model updates are performed with an incremental PCA algorithm. A spatial patch in the current frame is marked as foreground if either: 1) it is not well modeled by the PCA basis (i.e. large residual pixel error); or 2) the PCA coefficients extracted from the observed frame are different than those predicted from the previous frame (i.e. poor single-step prediction of the state variable).

The procedure now proposed makes two main contributions with respect to the state-of-the-art in dynamic background modeling [125, 126]. The first is
a background subtraction algorithm with the ability to classify the background according to the statistics of the whole spatio-temporal cube, instead of single-step predictions. This leads to foreground-background assignments that account for motion over multiple frames, and is more robust. The second is the generalized Stauffer-Grimson framework now proposed fully exploits the probabilistic modeling subjacent to the dynamic texture. This is unlike [125], which classifies pixels individually, or [126] which ignores the stochastic nature of the state variable. In result, the new algorithm is shown to substantially outperform the previous approaches.

VII.C  Stauffer-Grimson adaptive background modeling

In this section, we briefly review the static background subtraction method of SG [124]. The review emphasizes the insight that the background model can be updated using sufficient statistics. This suggests a natural generalization to dynamic scenes, which is introduced at the end of the section.

VII.C.1 Probabilistic model

The method of SG models each background pixel with an adaptive mixture of Gaussians. The probability of observing pixel $y$ is

$$p(y) = \sum_{j=1}^{K} \omega_j G(y, \mu_j, \Sigma_j),$$  \hspace{1cm} (VII.1)

where $K$ is the number of mixture components, $\omega_j$ the weight of each component, and $G(y, \mu, \Sigma)$ a multi-variate Gaussian distribution of mean $\mu$ and covariance $\Sigma$ (typically $\Sigma = \sigma I$). Given a video frame, this GMM is updated through an on-line approximation to the K-means algorithm.
VII.C.2 Online parameter estimation

A newly observed pixel value $y'$ is classified in a two step procedure. First, the Mahalanobis distance to each Gaussian component

$$d_j = \| y' - \mu_j \|_{\Sigma_j} = \sqrt{(y' - \mu_j)^T \Sigma_j^{-1} (y' - \mu_j)}$$  \quad (VII.2)$$

is computed. The closest component

$$k = \arg\min_j d_j$$  \quad (VII.3)$$
is then identified. The pixel is considered to match this component if the distance is within a threshold $\theta$, i.e. $d_k \leq \theta$ ([124] sets $\theta = 2.5$).

An on-line update of the model is performed after each pixel classification. If no match can be found, the model component of lowest weight $\omega_j$ is replaced with a new Gaussian of mean $y'$, an initially high variance, and a low initial weight. If $y'$ matches the $k$-th component, then its parameters are updated according to

$$\mu_k \leftarrow (1 - \alpha)\mu_k + \alpha y', \quad (VII.4)$$
$$\Sigma_k \leftarrow (1 - \alpha)\Sigma_k + \alpha(y' - \mu_k)(y' - \mu_k)^T, \quad (VII.5)$$

and the component weights adjusted with

$$w_j \leftarrow (1 - \beta)w_j + \beta \mathbb{I}(j = k), \forall j, \quad (VII.6)$$

where $\mathbb{I}(\cdot)$ is the indicator function, and re-normalized to sum to one.

It can be shown that the successive application of these on-line updates is equivalent to batch K-means estimation with exponentially decaying weights on the observations. This allows the mixture model to adjust to gradual non-stationary background changes (e.g. due to illumination). The adaptivity of the model is determined by the learning rates $\alpha$ and $\beta$: $\alpha$ controls how quickly each component adjusts to background changes, and $\beta$ controls the speed at which components become relevant$^1$. One appealing property of this on-line procedure is

$^1$[124] adaptively sets $\alpha = \beta G(y', \mu_k, \Sigma_k)$.  

that the existing components of the background model do not have to be destroyed as new pixel processes are added. This is useful when pixels revert to a previously-observed process. Because the associated mixture component still exists (although with a small weight), it can be quickly re-incorporated into the background model.

VII.C.3 Background detection

Because the on-line learning algorithm incorporates all pixel observations, the mixture model does not distinguish components that correspond to background from those associated with foreground objects, or simply due to noise. Due to this, background detection requires a list of “active” background components. In the method of SG [124], these are heuristically chosen as the Gaussians of most supporting evidence (high prior weight) and low variance. This is achieved by sorting components by decreasing ratio \( w_j/\sigma_j \), and selecting the first \( B \) that explain \( T \) percent of the probability (\( \sum_{j=1}^{B} w_j > T \)). Finally, the observed pixel is marked as background if its likelihood under these “active” background components is above a threshold, and foreground otherwise.

VII.C.4 Extending Stauffer-Grimson to other distributions

The choice of Gaussian mixture components is suitable when background pixels are static or change slowly over time. While this assumption sometimes holds (e.g. a fixed camera overlooking a roadway), there are numerous scenes where it does not (e.g. outdoors scenes involving water or vegetation). Such scenes require extensions of the SG model, using mixtures of component distributions that are appropriate for the particular background process. The main insight of the procedure proposed by SG is that, for Gaussian components, all parameter estimates can be obtained efficiently, through on-line updates of a small set of sufficient statistics.

The extension of the procedure to more complex distributions \( p(y|\Theta) \) requires the satisfaction of two conditions: 1) that the parameters \( \Theta \) of these dis-
Figure VII.2 Overview of generalized Stauffer-Grimson background modeling for dynamic textures. A video location is represented by a neighboring spatio-temporal volume $Y_{1:\tau}$. The location is marked as background if the log-likelihood of $Y_{1:\tau}$ under the active background component is above a threshold. Next, the mixture component with the largest log-likelihood of generating $Y_{1:\tau}$ is updated, if the log-likelihood is above threshold. Otherwise, a new component is learned, replacing the component of lowest prior probability.

Distributions can be estimated through a set of sufficient statistics $\zeta$, and 2) that these statistics can be computed through efficient on-line updates, such as those of (VII.4). The first condition is well known to be satisfied by any model in the exponential family [24]. Since the dynamic texture is a Gaussian model, and therefore in this family, it immediately satisfies this condition. For the second condition, we have already proposed an efficient online algorithm in Section II.D.3 for estimating the parameters. Hence, the dynamic texture model can be incorporated into the online mixture framework.

VII.D The generalized Stauffer-Grimson algorithm for dynamic textures

In this section, we introduce an adaptive background model based on the mixture of dynamic textures. This model is used to extend the background subtraction algorithm of [124] to dynamic scenes. An overview of the proposed algorithm is shown in Figure VII.2. Each video location is represented by a spatiotemporal neighborhood, centered at that location (in this work we use a $7 \times 7 \times 5$
The background scene is modeled as a mixture of $K$ dynamic textures, from which spatiotemporal volumes are drawn. The $j$-th dynamic texture is denoted (both its parameters and image statistics) by $\Theta_j$, and a prior weight $\omega_j$, s.t. $\sum_{j=1}^{K} \omega_j = 1$, is associated with each dynamic texture.

Given a spatiotemporal observation $Y_{1:\tau} \in \mathbb{R}^{m \times \tau}$ ($m = 49$ and $\tau = 5$ in all experiments reported), the location is marked as background if the log-likelihood of the observation under an “active” background component is above a threshold. The background model is then updated, using an on-line approximation to EM. As in SG, this consists of updating the mixture component with the largest log-likelihood of generating the observation $Y_{1:\tau}$, if the log-likelihood is above a second threshold. If not, a new dynamic texture component learned from $Y_{1:\tau}$ replaces the mixture component with lowest prior weight. In the remainder of the section, we discuss the two major components of the algorithm, background detection and on-line model updates.

VII.D.1 Background detection

The determination of whether a location belongs to the background requires the assignment of mixture components to background and foreground. We have already seen that the procedure proposed by SG [124] accomplishes this by heuristically ranking the mixture components. Support for multiple background components is crucial for SG because background colors can switch quickly (e.g. a flashing light). Under the dynamic texture mixture model, rapid changes in color and texture are modeled by the dynamic texture components themselves. It follows that multiple active background components are not necessary, and we simply select the component of largest prior,

$$i = \arg\max_j w_j,$$

as the “active” background component. A video location is marked as belonging to the background if the log-likelihood of the corresponding spatio-temporal volume
$Y_{1:\tau}$ under this mixture component is greater than a threshold $T$,

$$\log p(Y_{1:\tau}|\Theta_i) \geq T. \quad (VII.8)$$

Note that the log-likelihood can be rewritten in “innovation” form

$$\log p(Y_{1:\tau}; \Theta_i) = \log p(y_{1}; \Theta_i) + \sum_{t=2}^{\tau} \log p(y_{t}|Y_{1:t-1}; \Theta_i),$$

which can be efficiently computed with recourse to the Kalman filter (Section II.C).

**VII.D.2 On-line updating of the background model**

The background mixture model is learned with an on-line K-means algorithm. During training, an initial dynamic texture $\Theta_1$ is learned with the least-squares algorithm (Section II.D.2), and mixture weights are set to $\omega = [1, 0, \ldots, 0]$. Given a new spatio-temporal observation $Y_{1:\tau}$, the mixture parameters are updated as follows. First, the mixture component with the largest log-likelihood of having generated the observation

$$k = \arg\max_j \log p(Y_{1:\tau}; \Theta_j) \quad (VII.9)$$

is selected. If this log-likelihood is above the threshold $T$, i.e.

$$\log p(Y_{1:\tau}; \Theta_k) \geq T, \quad (VII.10)$$

the sufficient statistics of the k-th component are combined with the sufficient statistics \{\Phi', \phi', \varphi', \psi, \eta', \xi'\} derived from $Y_{1:\tau}$, in a manner similar to (VII.4),

$$\Phi \leftarrow (1 - \alpha)\Phi + \alpha\Phi', \quad \eta \leftarrow (1 - \alpha)\eta + \alpha\eta',$$
$$\phi \leftarrow (1 - \alpha)\phi + \alpha\phi', \quad \xi \leftarrow (1 - \alpha)\xi + \alpha\xi',$$
$$\psi \leftarrow (1 - \alpha)\psi + \alpha\psi', \quad \bar{y} \leftarrow (1 - \alpha)\bar{y} + \alpha\bar{y}'$$
$$\varphi \leftarrow (1 - \alpha)\varphi + \alpha\varphi'. \quad (VII.11)$$

As before, $\alpha$ is a learning rate which weighs the contribution of the new observation. Finally, the parameters of the mixture component are re-estimated with (II.71)
from the online least-squares algorithm (Section II.D.3), and prior weights are adjusted according to

$$w_j \leftarrow (1 - \beta)w_j + \beta \mathbb{I}(j = k), \forall j,$$

(VII.12)

(and normalized to sum to one). It can be shown that the successive application of the on-line estimation algorithm is equivalent to batch least-squares estimation with exponentially decaying weights on the observations. This allows the dynamic texture to adapt to slow background changes (e.g. lighting and shadows).

If (VII.10) does not hold, the component with smallest prior (i.e. \(i = \arg\min_j w_j\)) is replaced by a new dynamic texture learned from \(Y_{1:t}\). A regularization term \(\sigma I\) is added to the sufficient statistics \(\{\Phi, \phi, \varphi, \eta\}\), to guarantee a large initial variance (in \(Q, S,\) and \(R\)). As the component is updated with more observations, the influence of this regularization term vanishes. Finally, prior weights are adjusted according to

$$\omega_j \leftarrow (1 - \beta)\omega_j \mathbb{I}(j \neq i) + \beta \mathbb{I}(j = i), \forall j,$$

(VII.13)

and normalized to sum to one. The learning rate \(\beta\) adjusts the speed at which prior weights change, controlling how quickly a mixture component can become the “active” background component. The on-line background update algorithm is summarized in Algorithm 8.

When the dimension of the spatio-temporal volume, or the number of dynamic texture components, is large it may be impractical to compute and store the required sufficient statistics. In this case, the latter can be approximated by using a common PCA basis defined by \(\hat{\Phi}\) (see Appendix VII.H for details), which is updated with an incremental PCA algorithm. One disadvantage of approximating the covariance \(\Phi\) by its principal components is that some important directions of variation may not be captured (especially if the number of principal components is small). For example, a new PCA basis might not have strong enough support from a single video sample, but this support may increase considerably when aggregating
Algorithm 8 Online updating of the background model

**Input:** Spatio-temporal observation \( Y_{1:τ} \), dynamic texture components \( \{\Theta_j\}_{j=1}^K \), prior weights \( ω \), learning rates \( α \) and \( β \), threshold \( T \).

Find closest component: \( k = \arg\max_j \log p(Y_{1:τ}; Θ_j) \).

if \( \log p(Y_{1:τ}; Θ_k) \geq T \) then

- {Update component \( k \)}
  - Update the sufficient statistics of \( Θ_k \) with \( Y_{1:τ} \) using (VII.11).
  - Estimate the dynamic texture parameters \( Θ_k \) with (II.71).
  - Adjust priors (VII.12) and normalize.

else

- {Create new component}
  - Find smallest prior: \( k = \arg\min_j ω_j \).
  - Compute new sufficient statistics \( Θ_k \) from \( Y_{1:τ} \).
  - Estimate the parameters \( Θ_k \) with (II.71).
  - Adjust priors (VII.13) and normalize.

end if
over several iterations of the algorithm. In this case, the approximate algorithm will fail to include the new PCA basis, whereas the exact on-line algorithm will not.

VII.E Experiments

In this section, we present experiments on background subtraction using the adaptive background model based on dynamic textures. We present both quantitative and qualitative results for three aquatic video sequences, comparing results with other state-of-the-art models. A qualitative evaluation is also presented for several other challenging sequences.

VII.E.1 Videos with water backgrounds

The first set of experiments is based on three videos of objects moving in water. Bottle, from [125], displays a bottle floating in water (see Figure VII.4). The waves in the water move rapidly, pushing the bottle up the image. The sequence has dimensions $160 \times 120$, and the first 174 frames were used to train the background model, with the remaining 105 frames (which contain the actual bottle) being used for testing.

Boats1 and Boats2, depict boats moving in a harbor (see Figure VII.7 and Figure VII.8). The boats create a turbulent wake that dramatically changes the motion and appearance of the water. The sequences also contain a large perspective effect, which makes the water motion in the foreground significantly different from that in the background. The strong motion parallax of this scene makes these sequences very challenging for background subtraction. They have dimensions $180 \times 120 \times 300$. A separate sequence of 200 frames, containing just the baseline water motion, was used to train the background models.
VII.E.2 Experimental setup

Several background models, based on dynamic textures, were compared. The first represents each spatio-temporal volume as a simple non-adaptive dynamic texture, and is denoted by DT. The second and third models use the adaptive dynamic texture mixture with $K = 1$ and $K = 3$, and are denoted by DTM1 and DTM3, respectively. These models are updated with Algorithm 8 (when $K = 1$ no new components are added in the update phase). Finally, the last two models are based on the adaptive dynamic texture mixture (with $K = 1$ and $K = 3$) using approximate sufficient statistics (with $q = n$), and are denoted by DTM1x and DTM3x. In all cases, a spatio-temporal cube size of $7 \times 7 \times 5$ was adopted, and the state-space dimension of the dynamic textures was $n = 10$. The covariance matrix $R$ was also assumed diagonal. For the adaptive dynamic texture mixture, the learning rate parameters were set to $\alpha = 0.16$ and $\beta = 0.08$. Finally, when adding a new mixture component, the covariance regularization parameter was $\sigma = 10$.

We compared performance against several state-of-the-art background models. The first is the GMM of [124] with $\alpha = 0.01$. Specifically, we used the extension of [135], that automatically selects the number of mixture components, and is available from [136]. We have also implemented the models from [126] ($7 \times 7$ patches and $n = 10$) and [125] ($n = 10$). Finally, performance was compared against a simple background model, which represents $7 \times 7$ patches with $n = 10$ PCA components, and marks a patch as foreground if its reconstruction error is above a threshold.

All background models were evaluated by comparing foreground detections to ground-truth masks for foreground objects. Several examples of ground-truth are shown in Figure VII.4, Figure VII.7, and Figure VII.8. The foreground is displayed as white and the background as black. The gray region is the boundary between object and background. Due to the length of the sequences, and the fact that pixels along the object boundary contain a mixture of foreground and
background, it is both difficult and tedious to obtain pixel-accurate ground-truth masks. The gray boundary regions are ignored in the evaluation of foreground detection. Each background model was run for a large range of thresholds, and the true positive rate (TPR) and false positive rate (FPR), with respect to the ground-truth, were computed for each. The overall performance was measured by the area under the ROC curve (AUC). We also report the FPR at a specific high TPR, which we denote FPR-x (e.g. FPR-80 is the FPR at 80% TPR). Videos of the results are available in the supplemental [52].

Figure VII.3 ROC curves for foreground detection on Bottle video.

VII.E.3 Results on Bottle

We start by presenting results on Bottle. Since the water motion does not change significantly (i.e. its statistics are stationary), we only tested the non-adaptive dynamic texture (DT). The ROC curves of the different background models are shown in Figure VII.3, and the area under the ROC (AUC) is listed in Table VII.1. We also report the FPR at TPR of 0.80, denoted as FPR-80. The DT model performs significantly better than the GMM, with an AUC of 0.9985 for the former and 0.9229 for the latter. At 80% TPR, DT has a false-positive rate of 0.21%, while that of GMM is 2.36%. Note that although the DT model is non-adaptive,
it models the background motion accurately. On the other hand, despite its adaptive nature, GMM cannot cope with the stochasticity of the water motion. The PCA model also performs well, with an AUC of 0.9861, but is slightly inferior to DT. This suggests that modeling the water texture is sufficient for accurate performance, but the ability to further capture the texture dynamics gives the DT an additional performance boost.

Figure VII.4 shows the detected foreground regions at 80% TPR, for all background models. Note that the pixel-based methods (GMM and [125]) mark most of the reflection and shadow of the bottle as foreground, whereas the patch-based methods (DT, PCA, and [126]) classify it as background. Again, this
Table VII.1  Quantitative results on three water scenes for different background models.

<table>
<thead>
<tr>
<th>method</th>
<th>Bottle (AUC, FPR)</th>
<th>Boats1 (AUC, FPR)</th>
<th>Boats2 (AUC, FPR)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AUC</td>
<td>FPR-80</td>
<td>AUC</td>
</tr>
<tr>
<td>GMM</td>
<td>0.9229</td>
<td>0.2326</td>
<td>0.9516</td>
</tr>
<tr>
<td>PCA</td>
<td>0.9861</td>
<td>0.0041</td>
<td>0.9698</td>
</tr>
<tr>
<td>[126]</td>
<td>0.9912</td>
<td>0.0098</td>
<td>0.9061</td>
</tr>
<tr>
<td>[125]</td>
<td>0.9354</td>
<td>0.0226</td>
<td>0.9493</td>
</tr>
<tr>
<td>DT</td>
<td>0.9985</td>
<td>0.0021</td>
<td>0.9884</td>
</tr>
<tr>
<td>DTM1</td>
<td>-</td>
<td>-</td>
<td>0.9886</td>
</tr>
<tr>
<td>DTM3</td>
<td>-</td>
<td>-</td>
<td>0.9910</td>
</tr>
<tr>
<td>DTM1x</td>
<td>-</td>
<td>-</td>
<td>0.9881</td>
</tr>
<tr>
<td>DTM3x</td>
<td>-</td>
<td>-</td>
<td>0.9902</td>
</tr>
</tbody>
</table>

Figure VII.5  ROC curves for foreground detection on (a) Boats1, and (b) Boats2.

VII.E.4  Results on Boats1 and Boats2

The ROC curves for Boats1 and Boats2 are shown in Figure VII.5a and Figure VII.5b, and the results summarized in Table VII.1. We start by noting that the adaptive dynamic texture mixture with $K = 3$ (DTM3) outperforms all other methods at almost all levels of TPR. For example on Boats1 at 90% TPR, DTM3 has a FPR of 0.61%, whereas DTM1, DT, and GMM have FPR of 1.2%, 1.4% and 10.3%. On Boats2 at 55% TPR, DTM3 has a FPR of 0.88%, while
Figure VII.6  ROC curves for foreground detection with DTM and approximate DTM on (a) Boats1 and (b) Boats2.

DTM1, DT, and GMM have FPR of 3.18%, 7.11% and 2.61%. This performance difference is illustrated in Figure VII.7 and Figure VII.8, which show the detected foregrounds. As the boat traverses the scene, DTM3 models its wake with a new mixture component, quickly including it into the background. The adaptive model with a single dynamic texture (DTM1) takes much longer to adapt to the wake, because it contains a single mode.

While the methods that fully exploit the probabilistic representation of the dynamic texture tend to do well on these sequences, the remaining methods perform fairly poorly. GMM is able to adapt to the wake of the boat, but the overall foreground detection is very noisy, due to the constant wave motion. The method of [126] fails to detect the wake as part of the background, and has trouble with the stochastic nature of the shimmering waves in the bottom of the video frame. On Boats1, the simple PCA model outperforms GMM, [125], and [126]. Finally, the ROC plots comparing DTM with the approximate DTM are shown in Figure VII.6. The performance of DTM using the approximate sufficient statistics is similar to that of the standard DTM, albeit with a drop in performance on the difficult scene of Boats2. On this scene, the AUC drops from 0.9266 to 0.8955, suggesting that there is some loss in representative power when this approximation
is used.

Figure VII.7 Foreground detection results on Boats1 at 90% true-positive rate.

VII.E.5 Results on other video sequences

In addition to the above quantitative evaluation, we present qualitative results on five additional challenging sequences. The first is “zod2” from the PETS2005 coastal surveillance dataset [137], and is displayed in Figure VII.9. It was captured with a thermal camera, and contains a small rubber Zodiac boat, at a variety of scales, traveling in the ocean. The boat starts close to the camera, and travels directly away from it. In the distance, it turns towards the right and
Figure VII.8 Foreground detection results on Boats2 at 55% true-positive rate.
travels across the scene. At this point the boat is very small relative to its initial size (only occupies a few pixels). The boat wake is clearly visible when the boat is close but not visible when it is far away.

Foreground detection with the adaptive dynamic texture mixture (DTM3) is shown in Figure VII.9 (center). DTM3 marks the boat region as foreground, while ignoring the wake. The boat is also consistently marked as foreground, even when it is small relative to waves in other parts of the scene. This is in contrast to the GMM, which produces a significant amount of false detections. In particular, when the boat is small, the detected foreground region is very similar in shape and size to the false detections (e.g. see the last three rows of Figure VII.9). These types of errors make subsequent operations, such as object tracking, very difficult. On the other hand, DTM3 performs well regardless of the boat scale, maintaining a very low FPR throughout the sequence.

The second sequence is the beach scene from [126], and appears in Figure VII.10. The scene consists of two people walking on a beach, with part of the background composed of crashing waves. The waves have variable shape and intensity, making the background difficult to model. The foreground detections by DTM3 and GMM are shown in Figure VII.10. Again, DTM3 is able to adapt to the wave motion, while detecting the foreground objects. On the other hand, the crashing waves cause false alarms by the GMM.

The final three videos illustrate the performance of the adaptive mixture on scenes containing dynamic backgrounds other than water. Figure VII.11 shows a sequence of a helicopter traveling in dense smoke, along with the detected foreground using the simple dynamic texture (DT). The helicopter is marked as foreground by the model, while the smoke is marked as background. The next sequence, displayed in Figure VII.12, contains several people skiing down a mountain. The background includes falling snow flakes, and the scene is subject to an increasing amount of translation due to a camera pan. Figure VII.12 also shows the foreground detections by DTM3 and GMM. DTM3 successfully marks the
Figure VII.9  Foreground detection results on “zod2” from the PETS2005 coastal surveillance dataset [137].

Figure VII.10  Foreground detection results on the beach scene from [126].

Figure VII.11  Foreground detection results on Chopper video.
skiers as foreground, but exhibits some errors when there is a significant amount of panning (e.g. the last two frames). On the other hand, the foreground detected by GMM contains a significant amount of noise, due to both the falling snow and the camera pan, and fails to find the skiers when they are small (e.g. in the first two frames).

The final sequence, shown in Figure VII.13, shows a biker jumping in front of an explosion. In this scene the background is difficult to model because it changes significantly in a short period of time. Foreground detections by DTM3 are shown in Figure VII.13 (middle), illustrating the ability of the model to quickly adapt to an unseen background motion. The first frame shows the background before the explosion occurs. In the next frame, the beginning of the explosion is detected as foreground, because it is an unseen motion process. As the explosion propagates outward (3rd and 4th frames), the areas within the explosion are marked as background, because the model adapts to the new motion. Meanwhile, the biker is still marked as foreground. Finally, after sufficient exposure (the 5th and 6th frames), the explosion no longer registers as foreground, leaving only the biker in this category. Under the GMM model (Figure VII.13 bottom), the detected foreground again contains significant noise, indicating that the stochastic nature of the explosion is poorly modeled by the GMM pixel process.

VII.F Summary and discussion

In this work, we have introduced a generalization of the Stauffer-Grimson background subtraction algorithm. While the original algorithm restricts the background model to a Gaussian mixture, which is only suitable for static scenes, the generalization supports models with arbitrary component densities. The only restriction is that these densities can be summarized by sufficient statistics. We have applied the generalized SG model to the case where the component densities are dynamic textures, producing an adaptive background subtraction algorithm based
Figure VII.12 Foreground detection results on Skiing video.

Figure VII.13 Foreground detection results on CycleFire video.
on the mixture of dynamic textures, which is suitable for dynamic scenes. The performance of the new background subtraction algorithm was evaluated through extensive experiments, involving scenes with dynamic backgrounds. Substantial quantitative improvements over the state-of-the-art were demonstrated on aquatic scenes, where background dynamics are particularly challenging. For example, the proposed algorithm was shown to quickly incorporate boat wakes into the background, a task that proved too difficult for other state-of-the-art procedures. The efficacy of the proposed algorithm was also (qualitatively) demonstrated on other classes of dynamic backgrounds, including smoke, snow-fall, and fire. Finally, we note that the new background subtraction algorithm achieved high detection and low false-positive rates regardless of object scale, suggesting that it may be suitable for surveillance on dynamic scenes (e.g. harbors).

VII.G Acknowledgements

The authors thank Stan Sclaroff for the Bottle video from [125], Terry Boult for the zodiac video from PETS2005 [137], Gerald Dalley for the Beach video from [126], and Zoran Zivkovic for the GMM code [135]. The authors also thank Mulloy Morrow for capturing the Boats1 and Boats2 video.

The text of Chapter VII, in full, is based on the material as it appears in: A. B. Chan, V. Mahadevan, and N. Vasconcelos, “Generalized Stauffer-Grimson background subtraction for dynamic scenes”, submitted for publication to Machine Vision and Applications. The dissertation author was a primary researcher and an author of the cited material.

VII.H Appendix: Online estimation with approximate sufficient statistics

In this appendix, we derive the on-line estimation algorithm for dynamic textures (Sections II.D.3 and VII.D.2) using approximate sufficient statistics. If the
dimension of the video is large, then computing and storing the sufficient statistics may be impractical. In this case, the statistics can be approximated with the top \( q \) principal components of the image covariance \( \Phi \), i.e.

\[
\Phi \approx D\Phi_qD^T, \quad \phi \approx D\phi_qD^T, \quad \eta \approx D\eta_qD^T, \\
\psi \approx D\psi_qD^T, \quad \varphi \approx D\varphi_qD^T, \quad \xi \approx D\xi_q,
\]

(VII.14)

where \( D \in \mathbb{R}^{m \times q} \) is the matrix containing the \( q \) top PCA basis vectors of \( \Phi \). The approximate sufficient statistics, which are the projections of the sufficient statistics into the PCA basis of \( D \), are: \( \Phi_q \in \mathbb{R}^{q \times q} \) which is diagonal, \( \{\Phi_q, \phi_q, \eta_q, \psi_q, \varphi_q\} \in \mathbb{R}^{q \times q} \), and \( \xi_q \in \mathbb{R}^q \). The on-line estimation algorithm proceeds in three phases:

1) update the PCA basis \( D \), 2) update the approximate statistics, and 3) re-estimate the parameters of the dynamic texture. We will denote the previous PCA basis as \( D_{\text{old}} \), the current approximate statistics \( \{\Phi_q, \phi_q, \eta_q, \psi_q, \varphi_q, \xi_q\} \), and the new observation \( Y_{1:\tau} \).

In the first phase of the approximate algorithm, the PCA basis \( D_{\text{old}} \) must be updated with the new data \( Y_{1:\tau} \). An incremental PCA procedure is adopted, similar to [18]. Substituting the approximate image covariance into the update equation, given by (VII.11), yields

\[
\Phi \approx (1 - \alpha)D_{\text{old}}\Phi_qD_{\text{old}}^T + \alpha \left( \frac{1}{\tau}\tilde{Y}_{1:\tau}\tilde{Y}_{1:\tau}^T \right) = ZZ^T,
\]

(VII.15)

where

\[
Z = \left[ \sqrt{(1 - \alpha)D_{\text{old}}(\Phi_q)^\dagger}, \sqrt{\frac{\tau}{\tau}}\tilde{Y}_{1:\tau} \right].
\]

(VII.16)

Hence, the new PCA basis \( D \) can be computed from the SVD of \( Z = USV^T \),

\[
D = [u_1, \ldots, u_q], \quad (VII.17)
\]

\[
\Phi_q = \text{diag}([s_1^2, s_2^2, \cdots, s_q^2]), \quad (VII.18)
\]

where \( u_i \) are the columns of \( U \) corresponding to the \( q \) largest singular values \( \{s_1, \cdots, s_q\} \).
The second phase of the algorithm updates the approximate sufficient statistics with the statistics of the new video. Define the projection of the video $\tilde{Y}_{1,\tau}$ onto the basis of $D$ as

$$\hat{V}_{1,\tau} = [\hat{v}_1 \cdots \hat{v}_\tau] = D^T \tilde{Y}_{1,\tau}. \quad \text{(VII.19)}$$

Pre-multiplying (VII.14) by $D^T$ and post-multiplying by $D$, the estimates of the approximate statistics $\{\hat{\phi}_q', \hat{\varphi}_q', \hat{\psi}_q', \hat{\eta}_q', \hat{\xi}_q'\}$ of the new video $Y_{1,\tau}$ are

$$\hat{\phi}_q = D^T \hat{\phi}D = D^T \hat{Y}_{1,\tau-1} \hat{Y}_{1,\tau-1}^T = \hat{V}_{1,\tau-1} \hat{V}_{1,\tau-1}^T,$$

$$\hat{\varphi}_q = D^T \hat{\varphi}D = D^T \hat{Y}_{2,\tau} \hat{Y}_{2,\tau}^T = \hat{V}_{2,\tau} \hat{V}_{2,\tau}^T,$$

$$\hat{\psi}_q = D^T \hat{\psi}D = D^T \hat{Y}_{1,\tau-1} \hat{Y}_{1,\tau-1}^T = \hat{V}_{2,\tau} \hat{V}_{2,\tau-1}^T,$$

$$\hat{\eta}_q = D^T \hat{\eta}D = D^T \bar{y}_1 \bar{y}_1^T = \bar{v}_1 \bar{v}_1^T,$$

$$\hat{\xi}_q = D^T \hat{\xi} = D^T \bar{y}_1 = \bar{v}_1. \quad \text{(VII.20)}$$

The on-line update equations for the sufficient statistics (VII.11) have the form

$$\phi \leftarrow (1 - \alpha)\phi + \alpha\phi'. \quad \text{(VII.21)}$$

Substituting the approximate statistics into (VII.21), we have

$$D\hat{\phi}_q D^T \leftarrow (1 - \alpha)D_{old}\hat{\phi}_q D_{old}^T + \alpha D\hat{\phi}_q' D^T, \quad \hat{\phi}_q \leftarrow (1 - \alpha)(D^T D_{old})\hat{\phi}_q (D_{old}^T D) + \alpha\hat{\phi}_q'. \quad \text{(VII.22)}$$

Finally, defining $F = D^T D_{old}$, which transforms the approximate statistics from the old PCA basis into the new basis, the approximate statistics are updated according to

$$\hat{\phi}_q \leftarrow (1 - \alpha)F\hat{\phi}_q F^T + \alpha\hat{\phi}_q', \quad \hat{\eta}_q \leftarrow (1 - \alpha)F\hat{\eta}_q F^T + \alpha\hat{\eta}_q',$$

$$\hat{\varphi}_q \leftarrow (1 - \alpha)F\hat{\varphi}_q F^T + \alpha\hat{\varphi}_q', \quad \hat{\xi}_q \leftarrow (1 - \alpha)F\hat{\xi}_q + \alpha\hat{\xi}_q', \quad \bar{y} \leftarrow (1 - \alpha)\bar{y} + \alpha\bar{y}'. \quad \text{(VII.24)}$$

In the final phase, the dynamic texture parameters are estimated from the approximate sufficient statistics. From (II.58), the estimate of the observation matrix is

$$\hat{C} = \text{pca}(\hat{\Phi}, n) = \text{pca}(D\hat{\Phi}_q D^T, n) = DJ. \quad \text{(VII.25)}$$
where \( J = I_{q,n} \) is the \( q \times n \) identity matrix, which effectively selects the first \( n \) columns of \( D \). Next, substituting with (VII.14), we have

\[
\hat{C}^T \hat{\phi} \hat{C} \approx \hat{C}^T (D\hat{\phi}_q D^T) \hat{C} = J^T \hat{\phi}_q J,
\]

\[
\hat{C}^T \hat{\psi} \hat{C} \approx \hat{C}^T (D\hat{\psi}_q D^T) \hat{C} = J^T \hat{\psi}_q J,
\]

\[
\hat{C}^T \hat{\eta} \hat{C} \approx \hat{C}^T (D\hat{\eta}_q D^T) \hat{C} = J^T \hat{\eta}_q J,
\]

\[
\hat{C}^T \hat{\xi} \approx \hat{C}^T (D\hat{\xi}_q) = J^T \hat{\xi}_q,
\]

(VII.26)

where \( J^T \phi J \) selects the top-left \( n \times n \) sub-matrix of \( \phi \). Hence substituting (VII.26) into (II.71), the approximate parameter estimates are

\[
\hat{A} \approx (J^T \hat{\psi}_q J) (J^T \hat{\phi}_q J)^{-1}, \quad \hat{\mu} \approx J^T \hat{\xi}_q,
\]

(VII.27)

\[
\hat{Q} \approx \frac{1}{\tau^2} (J^T \hat{\phi}_q J - \hat{A} J^T \hat{\psi}_q J), \quad \hat{S} \approx (J^T \hat{\eta}_q J) - \hat{\mu} \hat{\mu}^T.
\]

Finally, the covariance of the observation noise in (II.71) can be written

\[
\hat{R} = (I - \hat{C} \hat{C}^T) \hat{\Phi} (I - \hat{C} \hat{C}^T) \approx (I - \hat{C} \hat{C}^T) ZZ^T (I - \hat{C} \hat{C}^T).
\]

(VII.28)
Chapter VIII

Application: People counting without people models or tracking
VIII.A Introduction

There is currently a great interest in vision technology for monitoring all types of environments. This could have many goals, e.g. security, resource management, or advertising. Yet, the deployment of vision technology is invariably met with skepticism by society at large, given the perception that it could be used to infringe on the individuals’ privacy rights. This tension is common in all areas of data-mining [138, 139], but becomes an especially acute problem for computer vision for two reasons: 1) the perception of compromised privacy is particularly strong for technology which, by default, keeps a visual record of people’s actions; 2) the current approaches to vision-based monitoring are usually based on object tracking or image primitives, such as object silhouettes or blobs, which imply some attempt to “identify” or “single out” the individual.

From the laymen’s point of view, there are many problems in environment monitoring that can be solved without explicit tracking of individuals. These are problems where all the information required to perform the task can be gathered by analyzing the environment holistically: e.g. monitoring of traffic flows, detection of disturbances in public spaces, detection of speeding on highways, or estimation of the size of moving crowds. By definition, these tasks are based on either properties of 1) the “crowd” as a whole, or 2) an individual’s “deviation” from the crowd. In both cases, to accomplish the task it should suffice to build good models for the patterns of crowd behavior. Events could then be detected as variations in these patterns, and abnormal individual actions could be detected as outliers with respect to the crowd behavior. This would preserve the individual’s identity until there is good reason to do otherwise.

In this chapter, we introduce a new formulation for surveillance technology, which is averse to individual tracking and, consequently, privacy preserving. We illustrate this new formulation with the problem of pedestrian counting. This is a canonical example of a problem that vision technology addresses with pri-
Figure VIII.1 Examples of a low-resolution scene containing a sizable crowd with inhomogeneous dynamics, due to pedestrian motion in different directions.

Privacy invasive methods: detect the people in the scene [140, 141, 142, 143, 144], track them over time [145, 146, 147], and count the number of tracks. While a number of methods that do not require explicit detection or tracking have been previously proposed [148, 149, 150, 151, 152, 153, 154], they have not fully established the viability of the privacy-preserving approach. This has a multitude of reasons: from limited applications to indoor environments with controlled lighting (e.g. subway platforms) [148, 149, 150, 151, 153]; to ignoring the crowd dynamics (i.e. treating people moving in different directions as the same) [148, 149, 150, 151, 152, 154]; to assumptions of homogeneous crowd density (i.e. spacing between people) [153]; to measuring a surrogate of the crowd size (e.g. crowd density or percent crowding) [148, 149, 153]; to questionable scalability to scenes involving more than a few people [154]; to limited experimental validation of the proposed algorithms [148, 149, 150, 152, 153].

Unlike these proposals, we show that there is in fact no need for pedestrian detection, object tracking, or object-based image primitives to accomplish the pedestrian counting goal, even when the crowd is sizable and inhomogeneous, e.g. has sub-components with different dynamics, as illustrated in Figure VIII.1. In fact, we argue that, when considered under the constraints of privacy-preserving monitoring, the problem actually appears to become simpler. We simply develop methods for segmenting the crowd into the sub-parts of interest (e.g. groups of people moving in different directions) and estimate the number of people by analyzing holistic properties of each component. This is shown to be quite robust
and accurate.

The contributions of this chapter are three-fold. First, we present a privacy-preserving vision system for estimating the size of inhomogeneous crowds that does not depend on object detection or feature tracking. The system is also privacy-preserving in the sense that it can be implemented with hardware that does not produce a visual record of the people in the scene, i.e. with special-purpose cameras that output low-level features (e.g. segmentations, edges, and texture). Second, we validate the system quantitatively on two large datasets of pedestrian video. Third, we demonstrate its robustness by presenting results on two hours of video. To our knowledge, this is the first privacy-preserving pedestrian counting system that accounts for multiple pedestrian flows, and successfully operates continuously in an outdoors, unconstrained, environment for such time periods. The remainder of the chapter is organized as follows. In Section VIII.B we review related work in crowd counting. In Section VIII.C, we introduce a crowd counting system based on motion segmentation and Gaussian processes. Finally, we present the pedestrian database and experimental results in Sections VIII.D and VIII.E.

VIII.B Related work

The taxonomy of crowd counting algorithms consists of three paradigms: 1) pedestrian detection, 2) visual feature trajectory clustering, and 3) feature-based regression. Pedestrian detection algorithms are based on boosting appearance and motion features [140], Bayesian model-based segmentation [141], histogram-of-gradients [155], or integrated top-down and bottom-up processing [142]. Because they detect whole pedestrians, these methods tend to suffer in very crowded scenes with significant occlusion, which has been addressed to some extent by adopting part-based detectors [143, 144, 156].

The second paradigm counts people by identifying and tracking visual features over time. The feature trajectories that exhibit coherent motion are clus-
Figure VIII.2 Correspondence between crowd size and two simple features: (left) segmentation area, and (right) the number of edge pixels in the segmented region. The Gaussian process (GP) regression function is also plotted with the two standard deviations error bars (gray area).

Feature-based regression for crowd counting was first applied to subway platform monitoring. These methods typically work by: 1) subtracting the background; 2) measuring various features of the foreground pixels, such as total area [148, 149, 151], edge count [149, 150, 151], or texture [153]; and 3) estimating the crowd density or crowd count by a regression function, e.g. linear [148, 151], piece-wise linear [150], or neural networks [149, 153]. In recent years, feature-based regression has also been applied to outdoor scenes. For example, [152] applies neural networks to the histograms of foreground segment areas and edge orientations. [154] estimates the number of people in each foreground segment by matching its shape to a database containing the silhouettes of possible people configurations, but is only applicable when the number of people in each segment is small (empirically, less than 6).

VIII.C Privacy preserving crowd counting

Feature-based regression for crowd counting was first applied to subway platform monitoring. These methods typically work by: 1) subtracting the background; 2) measuring various features of the foreground pixels, such as total area [148, 149, 151], edge count [149, 150, 151], or texture [153]; and 3) estimating the crowd density or crowd count by a regression function, e.g. linear [148, 151], piece-wise linear [150], or neural networks [149, 153]. In recent years, feature-based regression has also been applied to outdoor scenes. For example, [152] applies neural networks to the histograms of foreground segment areas and edge orientations. [154] estimates the number of people in each foreground segment by matching its shape to a database containing the silhouettes of possible people configurations, but is only applicable when the number of people in each segment is small (empirically, less than 6).

VIII.C Privacy preserving crowd counting

Figure VIII.1 shows examples of a crowded scene on a pedestrian walkway. The goal of the proposed system is to estimate the number of people moving in
Figure VIII.3 Overview of the crowd counting system: the scene is segmented into crowds moving in different directions. Features, which are normalized to account for perspective, are extracted from each segment, and the number of people in each segment is estimated with a Gaussian process.

Each direction, in a privacy-preserving manner. Given a segmentation into the two sub-components of the crowd, it is shown that crowd size can indeed be estimated from low-level features extracted from each crowd segment. For example, as shown in Figure VIII.2, simple features such as the segmentation area or the number of edge pixels within the segmented region are approximately linear in the crowd size.

An outline of the crowd counting system appears in Figure VIII.3. The video is segmented into crowd regions moving in different directions, using a mixture of dynamic textures. For each crowd segment, various features are extracted, while applying a perspective map to weight each image location according to its approximate size in the real scene. Finally, the number of people per segment is estimated with Gaussian process regression. The remainder of this section describes each of these system components.

VIII.C.1 Crowd segmentation

We use the mixture of dynamic textures (Chapter IV) to segment the crowds moving in different directions. The video is represented as collection of spatio-temporal patches, \( (7 \times 7 \times 20 \text{ and } 13 \times 13 \times 10 \text{ patches were used}) \), which are modeled as independent samples from a mixture of dynamic textures. The mixture
model is learned with the expectation-maximization (EM) algorithm (again, see Chapter IV). Video locations are then scanned sequentially, a patch is extracted at each location, and assigned to the mixture component of largest posterior probability. The location is declared to belong to the segmentation region associated with that component. For long sequences, where characteristic motions are not expected to change significantly, the computational cost of the segmentation can be reduced by learning the mixture model from a subset of the video (e.g. a representative clip). The remaining video can then be segmented by computing the posterior assignments as before. This procedure tends to work well in practice, and was used to segment a two full hour of video. The resulting segmentations are illustrated in Figure VIII.12, Figure VIII.14, Figure VIII.18, and Figure VIII.20.

VIII.C.2 Perspective normalization

Before extracting features from the video segments, it is important to consider the effects of perspective. Because objects closer to the camera appear larger, any feature extracted from a foreground object will account for a smaller portion of the object than one extracted from an object farther away. This makes it important to normalize the features for perspective. One possibility is to weight each pixel according to a perspective normalization map. The pixel weight is based on the expected depth of the object which generated the pixel, with larger weights given to far objects.

In this work, we approximate the perspective map by linearly interpolat-
ing between the two extremes of the scene. A ground plane is first marked, as in Figure VIII.4a, and the distances \(|\overline{ab}|\) and \(|\overline{cd}|\) are measured\(^1\). Next, a reference pedestrian is selected, and the heights \(h_1\) and \(h_2\) are measured when the center of the person is on \(\overline{ab}\) and \(\overline{cd}\) (see Figure VIII.4a and Figure VIII.4b). The pixels on \(\overline{ab}\) are given a weight of 1, and the pixels on \(\overline{cd}\) a weight of \(\frac{h_1|\overline{ab}|}{h_2|\overline{cd}|}\). Finally, the remaining pixel weights are computed by interpolating linearly between the two lines. Figure VIII.4c shows the perspective map of the scene using the above procedure. In this case, objects on the front-line \(\overline{ab}\) are approximately 2.4 times bigger than objects on the back-line \(\overline{cd}\). Finally, for features based on area (e.g. segmentation area), the weights are applied to each pixel. For features based on edges (e.g. edge histogram), the square-roots of the weights are used.

### VIII.C.3 Feature extraction

Ideally, features such as segmentation area or number of edges should vary linearly with the number of people in the scene \([151, 148]\). Figure VIII.2 plots the segmentation area versus the crowd size. While the overall trend is indeed linear, there exist local non-linearities that arise from a variety of factors, including occlusion, segmentation errors, and pedestrian configuration (e.g. spacing within a segment). To model these non-linearities, we extract an additional 28 features from each crowd segment.

**Segment features:** These features capture segment shape and size.

- *Area* – total number of pixels in the segment.
- *Perimeter* – total number of pixels on the segment perimeter, computed with morphological operators.
- *Perimeter edge orientation* – orientation histogram of the segment perimeter. The orientations are quantized into 6 bins, and opposite orientations (180°)

\(^1\)Here we assume that the horizontal ground plane is parallel to the horizontal axis of the image, but the procedure can be generalized if not.
apart) are considered equal. The orientation of each edge pixel is computed by finding the maximum response to a set of oriented Gaussian filters at that point.

- *Perimeter-area ratio* – ratio between the segment perimeter and area, which measures the complexity of the segment shape. Segments of high ratio contain bumps in their perimeter, which may be indicative of the number of people contained within.

- *“blob” count* – the number of connected components in the segment.

**Internal edge features:** The edges contained in a crowd segment are a strong clue about the number of people in the segment [152, 151]. A Canny edge detector [157] is applied to the entire image, the edge image is masked by the crowd segmentation, and the following features are computed:

- *Total edge pixels* – total number of edge pixels contained in the segment.

- *Edge orientation* – histogram of the edge orientations in the segment, generated in the same way as the perimeter edge histogram (also using 6 bins).

- *Minkowski dimension* – the Minkowski fractal dimension of the edges in the segment, which estimates their degree of “space-filling” (see [158] for more details).

**Texture features:** Texture features, based on the gray-level co-occurrence matrix (GLCM), were used in [153] to classify image patches into 5 classes of crowd density (very low, low, moderate, high, and very high). In this work, we adopt a similar set of measurements for counting the number of pedestrians in each segment. The image is quantized into 8 gray levels, and the 2nd-order joint conditional probability density function $f(i, j|d, \theta)$ is estimated for distance $d = 1$ and angles $\theta = \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$. The following texture properties are computed:

- *Homogeneity:* $S_h(d, \theta) = \sum_{i,j} \frac{f(i,j|d,\theta)}{1+(i-j)^2}$
• **Energy**: \( S_g(d, \theta) = \sum_{i,j} f(i, j|d, \theta)^2 \)

• **Entropy**: \( S_p(d, \theta) = \sum_{i,j} f(i, j|d, \theta) \log f(i, j|d, \theta) \)

resulting in a total of 12 texture features.

### VIII.C.4 Gaussian process regression

A Gaussian process (GP) [159] is used to regress feature vectors to the number of people per segment. The GP defines a distribution over functions, which is “pinned down” at the training points. The classes of functions that the GP can model is dependent on the kernel function used. For example, Bayesian linear regression uses a linear kernel

\[
k(x_p, x_q) = \alpha_1 (x_p^T x_q + 1) + \alpha_2 \delta(x_p, x_q)
\]  

(VIII.1)

where \( \alpha = \{\alpha_1, \alpha_2\} \) are the hyperparameters that weight the linear term (first term) and the observation noise (second term). For the task of pedestrian counting, we note that the dominant trend of many of the features is linear (e.g. segment area), as shown in Figure VIII.2, with some local non-linearities due to occlusions and segmentation errors. However, the inclusion of other features may make the dominant trend non-linear. Hence, to capture both the dominant trend and the local non-linearities, we combine two squared-exponential (RBF) kernels, i.e.

\[
k(x_p, x_q) = \alpha_1 e^{-\frac{\|x_p-x_q\|^2}{2\beta_1^2}} + \alpha_2 e^{-\frac{\|x_p-x_q\|^2}{2\beta_2^2}} + \alpha_3 \delta(x_p, x_q)
\]  

(VIII.2)

with hyperparameters \( \alpha = \{\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3\} \). The first RBF component models the overall trend (with a large length scale \( \beta_1 \)) and the second RBF component models the local non-linearities (with a small length scale \( \beta_2 \)), while the third term models observation noise.

Given a set of function inputs and outputs \( \{x, y\} \), the kernel hyperparameters are determined by maximizing the marginal likelihood \( p(y|x, \alpha) \), which marginalizes over the class of regression functions (see Chapter 5 of [159] for more
details). Figure VIII.2 shows an example of GP regression for segmentation area. Note that the regression function captures both the dominant trend and the local non-linearities. Finally, while the same feature set is used throughout the system, a different regressor is learned for each direction of crowd motion because the appearance changes with the traveling direction.

VIII.D Pedestrian database

In this section, we describe the pedestrian databases used in the experiments. Two hours of video was collected from two viewpoints overlooking a pedestrian walkway at UCSD using a stationary digital camcorder. The first viewpoint, shown in Figure VIII.5 (left), is an oblique view of a walkway, and contains a large number of people. The second viewpoint, shown in Figure VIII.5 (right), is a side-view of a walkway, and contains fewer people. We refer to these two viewpoints, and corresponding databases, as Peds1 and Peds2, respectively. The original video was captured at 30 fps with a frame size of 740 × 480, and was later downsampled to 238 × 158 and 10 fps. The first 4000 frames (400 seconds) of each video were selected for ground-truth annotation.

Figure VIII.5 Ground-truth pedestrian annotations for: (left) Peds1 database: red and green tracks are people moving away from, and towards the camera; (right) Peds2 database: red and green tracks are people walking right or left, while cyan and yellow tracks are fast objects moving right or left. The ROI for the experiments is highlighted.
Table VIII.1 Properties of the Peds1 database

<table>
<thead>
<tr>
<th>frame size</th>
<th>238 × 158</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>400 sec (10 fps)</td>
</tr>
<tr>
<td>number of frames</td>
<td>4000 frames</td>
</tr>
<tr>
<td>training/test size</td>
<td>1200/2800 frames</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class</th>
<th>away</th>
<th>towards</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td># of unique people</td>
<td>166</td>
<td>169</td>
<td>335</td>
</tr>
<tr>
<td>max # in frame</td>
<td>33</td>
<td>20</td>
<td>46</td>
</tr>
<tr>
<td>min # in frame</td>
<td>3</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>total # of people</td>
<td>45,931</td>
<td>47,685</td>
<td>93,616</td>
</tr>
<tr>
<td># of training people</td>
<td>15,085</td>
<td>16,340</td>
<td>31,425</td>
</tr>
<tr>
<td># of test people</td>
<td>30,846</td>
<td>31,345</td>
<td>62,191</td>
</tr>
</tbody>
</table>

Table VIII.2 Properties of the Peds2 database.

<table>
<thead>
<tr>
<th>frame size</th>
<th>238 × 158</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>400 sec (10 fps)</td>
</tr>
<tr>
<td>number of frames</td>
<td>4000 frames</td>
</tr>
<tr>
<td>training/test size</td>
<td>1000/3000 frames</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class</th>
<th>right-slow</th>
<th>left-slow</th>
<th>right-fast</th>
<th>left-fast</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td># of unique people</td>
<td>196</td>
<td>140</td>
<td>6</td>
<td>2</td>
<td>344</td>
</tr>
<tr>
<td>max # in frame</td>
<td>11</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>min # in frame</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>total # of people</td>
<td>12,623</td>
<td>9,223</td>
<td>145</td>
<td>49</td>
<td>22,040</td>
</tr>
<tr>
<td># of training people</td>
<td>3,816</td>
<td>2,526</td>
<td>45</td>
<td>25</td>
<td>6,412</td>
</tr>
<tr>
<td># of test people</td>
<td>8,807</td>
<td>6,697</td>
<td>100</td>
<td>24</td>
<td>15,628</td>
</tr>
</tbody>
</table>
A region-of-interest (ROI) was selected on the walkway (see Figure VIII.5), and the traveling direction (motion class) and visible center of each pedestrian was annotated every five frames. Pedestrian locations in the remaining frames were estimated with linear interpolation. The Peds1 database was annotated with two motion classes: “away” from or “towards” the camera. For the Peds2 database, the classes were also split into fast and slow motion, resulting in four motion classes: “right-slow”, “left-slow”, “right-fast”, and “left-fast”. An example annotation is shown in Figure VIII.5 for each of the databases. Note that the ground-truth pedestrian locations are not required to train the crowd-counting system, but necessary to test performance of the motion segmentation algorithm. The Peds1 dataset contains 335 unique people, with a minimum of 11 and a maximum of 46 appearing at a time, and a total of 93,616 pedestrian instances (see Table VIII.1 for a summary). The Peds2 dataset contains 344 unique people, with a minimum of 0 and a maximum of 15 appearing at the same time, and a total of 20,040 pedestrian instances (see Table VIII.2 for a summary). Finally, Figure VIII.5 presents the ground-truth pedestrian count over time, for both databases.

Each database was split into a training set, for learning the GP, and a test set, for validation. For Peds1, the training set contains 1200 frames, between frame 1400 and 2600, with the remaining 2800 frames held out for testing. For Peds2, the training set contains 1000 frames, between frame 1500 and 2500, with the remaining 3000 frames held out for testing. Note that these splits test the ability of the crowd-counting system to extrapolate beyond the training set. In contrast, spacing the training set evenly throughout the dataset would only test the ability to interpolate between the training points, which provides little insight into generalization ability and robustness. The properties of the pedestrian dataset are summarized in Table VIII.1 and Table VIII.2.

\[1^2\text{Bicyclists and skateboarders were treated as normal pedestrians.}\]
Figure VIII.6  Ground-truth pedestrian counts for: (a) Peds1 database; (b, c) Peds2 database.
VIII.E Experimental evaluation

Success of the crowd counting system depends on effective crowd segmentation. Hence, we first test the segmentation algorithm, and then present crowd counting results.

VIII.E.1 Motion segmentation results

The mixture of dynamic textures was used to segment the crowd in Peds1 according to motion: people moving towards the camera, and people moving away. Segmentation examples can be seen in Figure VIII.12. The segmentation was validated with an ROC curve based on the ground-truth pedestrian locations. In each frame, a true positive is recorded if the ground-truth location of a person is within the correct motion segment, and a false positive is recorded otherwise. The true positive and false positive rates (TPR and FPR) are computed over the first 2000 frames of Peds1, and an ROC curve was generated from the TPR and FPR for dilations and erosions of the segmentation with variable size disks.

The ROC curve produced by the mixture of dynamic textures (DTM) is shown in Figure VIII.7a. For comparison, the scene was also segmented with normalized cuts and motion-profiles [84], which is denoted by NCuts. At the operating point of the segmentation algorithms (i.e. no morphological post-processing), DTM achieves a high TPR of 0.936, at a low FPR of 0.036. NCuts achieves a lower TPR (0.890) at a higher FPR (0.260). In addition, DTM has a larger area under the curve (AUC) than NCuts (0.9727 versus 0.9545). Figure VIII.7b shows the breakdown of the AUC for different crowd sizes (number of people in the scene), and Figure VIII.7c shows the frequency of each crowd size. DTM outperforms NCuts for almost all crowd sizes. In addition, the performance only degrades slightly as the crowd size increases. These results validate the DTM as a robust segmentation algorithm for these types of crowded scenes.
Figure VIII.7 Crowd motion segmentation on Peds1: a) ROC curve for the mixture of dynamic textures (DTM) and normalized cuts (NCuts). The circle indicates the operating point of the algorithm; b) area under ROC for different crowd sizes; c) frequency of crowd sizes.
Table VIII.3 Comparison of feature sets and regression methods on Peds1.

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Regression</th>
<th>MSE away</th>
<th>MSE towards</th>
<th>err away</th>
<th>err towards</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>GP-lin</td>
<td>4.9800</td>
<td>2.6961</td>
<td>1.6643</td>
<td>1.3332</td>
</tr>
<tr>
<td>segm</td>
<td>GP-lin</td>
<td>4.4418</td>
<td>2.8246</td>
<td>1.6032</td>
<td>1.3289</td>
</tr>
<tr>
<td>edge</td>
<td>GP-lin</td>
<td>4.6546</td>
<td>2.2804</td>
<td>1.6618</td>
<td>1.1596</td>
</tr>
<tr>
<td>segm+edge</td>
<td>GP-lin</td>
<td>4.0000</td>
<td>2.1150</td>
<td>1.5693</td>
<td>1.1243</td>
</tr>
<tr>
<td>all</td>
<td>GP-lin</td>
<td>3.2657</td>
<td>2.6821</td>
<td>1.4371</td>
<td>1.2771</td>
</tr>
<tr>
<td>all</td>
<td>GP-RR</td>
<td><strong>2.9768</strong></td>
<td><strong>2.0232</strong></td>
<td><strong>1.4068</strong></td>
<td><strong>1.0896</strong></td>
</tr>
<tr>
<td>all</td>
<td>lin</td>
<td>3.3118</td>
<td>2.8643</td>
<td>1.4439</td>
<td>1.3250</td>
</tr>
<tr>
<td>all</td>
<td>rob-lin</td>
<td>3.3871</td>
<td>2.9579</td>
<td>1.4593</td>
<td>1.3471</td>
</tr>
<tr>
<td>[151]</td>
<td>GP-lin</td>
<td>4.4754</td>
<td>2.0371</td>
<td>1.7039</td>
<td>1.0771</td>
</tr>
<tr>
<td>[152]</td>
<td>GP-lin</td>
<td><strong>4.8364</strong></td>
<td><strong>1.9336</strong></td>
<td><strong>1.6993</strong></td>
<td><strong>1.0464</strong></td>
</tr>
</tbody>
</table>

**VIII.E.2 Crowd counting results on Peds1**

The crowd counting system was trained on the 1200 training frames from Peds1, and tested on the remaining 2800 of the video. The GP output was rounded to the nearest integer to produce a crowd count, and both the mean-squared-error (MSE) and the absolute error between this estimate and the ground-truth were recorded. For comparison, we trained the system with different subsets of the features (only segment area, the segment features, and the segment and edge features). We also tested a variety of regression functions and kernels, including: (GP-lin) GP with linear kernel, (GP-RR) GP with RBF-RBF kernel, (lin) linear regression, and (rob-lin) robust linear regression. Finally, we compared performance against the feature sets of [152] (segment size histogram and edge orientation histogram) and [151] (segment area and total edges) using GP-lin, which gave the best performance among the various regression algorithms.

Table VIII.3 shows the error rates for the two crowd directions, under the different feature representations. We first examine the performance on the “away” crowd, which contains more people. Using only the area feature performs the worst, with an MSE of 4.980 for GP-lin regression, and performance improves steadily as other features are added. Using all the segment features improves the
MSE to 4.442, and using both the segment and edge features further improves to 4.000. Finally, using all features (i.e. adding the texture features) performs best, with an MSE of 3.266. This demonstrates the informativeness of the different feature subsets when there are large crowds: the segment features provide a coarse linear estimate, which is refined by the edge and texture features accounting for various non-linearities. On the other hand, using the texture features hinders the performance on the “towards” crowd, which contains fewer people (MSE increases from 2.115 to 2.682). This indicates that the texture features are only informative when there are many people in the crowd segment. Adopting the RBF-RBF kernel (GP-RR) improves the performance on both crowd segments, with the MSE dropping from 3.266 to 2.977 for the “away” crowd, and 2.682 to 2.023 for the “towards” crowd. Finally, compared to [152, 151], the full feature set performs better on the “away” crowd (MSE of 2.977 versus 4.475 and 4.836), but performs similarly on the “towards” crowd (MSE of 2.023 vs 2.037 and 1.934). Again, this indicates that the texture features are more informative when there are more people in the crowd segment.

Figure VIII.8 shows the crowd count estimates (using all features and GP-RR) as a function of time, for the two crowd directions. The estimates track the ground-truth well in most of the test frames. The overestimate of the size of the “away” crowd in frames 180-300 is caused by two bicyclists traveling quickly through the scene, as shown in the third image of Figure VIII.12. Figure VIII.9 shows the cumulative error, i.e. the frequency with which the counting error is below a particular number of people. The count is within 3 people of the ground-truth 97% of the time for the “away” crowd, and within 3 people 99% of the time for the “towards” crowd. On average, the count estimate is within 1.41 and 1.09 (“away” and “towards”, respectively) of the ground-truth. Comparing with the features set from [152] (Kong), the cumulative error is worse than GP-RR on the “away” crowd (within 3 people only 88% of the time, versus 97%), but performs similarly on the “towards” crowd. In addition, Figure VIII.10 shows the
average error for different crowd sizes. For GP-RR, using all features, the error is relatively constant regardless of the crowd size. On the other hand, for [152] the error increases as the crowd size increases. This suggests that the system using the entire feature set and GP-RR regression is robust and accurate enough for monitoring pedestrian traffic over long time-periods. Figure VIII.12 shows the original image, segmentation, and crowd estimates for several frames in the test set. A video is also available in the supplemental [52].

In an additional experiment, we measured the test error while varying the size of the training set, by picking a subset of the original training set. Figure VIII.11 plots the test error versus the training set size, for different regression functions. The GP-lin and GP-RR are more robust when there are fewer training examples, compared with linear and robust linear regression. This indicates that, in practice, a system could be trained with much fewer examples, which reduces the number of images that need to be annotated by hand.

![Graph showing test error versus training set size](image)

Figure VIII.8 Crowd counting results on Peds1 over both the training and test sets for: (left) “away” crowd, and (right) “toward” crowd. The gray bars show the two standard-deviations error bars of the GP.

Finally, we trained GP-RR on 2000 frames of the pedestrian dataset (every other frame), and ran the system on the remaining 50 minutes of captured
Figure VIII.9  Cumulative error on Peds1 for: (left) “away” crowd, and (right) “toward” crowd.

Figure VIII.10  Error splits for different crowd sizes on Peds1 for: (left) “away” crowd, and (right) “towards” crowd.

Figure VIII.11  Test error versus training size for: (left) “away” crowd, and (right) “towards” crowd, for different regression algorithms.
Figure VIII.12 Crowd counting examples from Peds1: The red and green segments are the “away” and “towards” crowds. The estimated crowd count for each segment is in the top-left, with the (standard-deviation of the GP) and the [ground-truth]. The ROI is also highlighted.
video. The resulting crowd estimates are shown in Figure VIII.13, while Figure VIII.14 shows several example outputs of the system (the video is also available from [52]). Qualitatively, the system tracks the changes in pedestrian traffic fairly well. Most errors tend to occur when there are very few people (less than two) in the scene. These errors are reasonable, considering that there are no training examples with such few people, and the problem could be fixed by simply adding training examples of such cases. Note that the GP signals its lack of confidence in these estimates, by assigning them larger error bars.

A more challenging set of errors occur when bicycles, skateboarders, and golf carts travel quickly on the walkway. Again, these errors are reasonable, since there are very few examples of fast moving bicycles and no examples of golf carts in the training set. These cases could be handled by adding more mixture components to the segmentation algorithm, which would label fast moving objects as different classes. Another GP could then be trained to count the number of fast moving vehicles in the scene. Another possibility would be to simply identify these objects as outliers, based on the posterior assignment probabilities of the segmentation stage. Any of these possibilities would require larger training sets, with a richer representation of the outliers.
Figure VIII.13  Count estimates on 55 minutes of Peds1: (top) “away” crowd; (bottom) “towards” crowd. The shaded bars indicate periods when the GP model had low confidence ($\sigma > 3$).
Figure VIII.14  Example counting results on 55 minutes of Peds1. The counts are in the top-left, with the (standard-deviation).
VIII.E.3 Crowd counting results on Peds2

The crowd counting system was trained on the 1000 training frames from Peds2, and tested on the remaining 3000 of the video. Since Peds2 contains smaller crowds, we use the segment and edge features with GP-lin regression. Table VIII.4 shows the error rates for the four crowd segments, “right-slow”, “left-slow”, “right-fast”, and “left-fast”, under the different feature representations. Again, the performance improves as more features are used. In particular, the MSE decreases from 0.823/0.624 (“right-slow”/“left-slow”) when using just the area feature, to 0.685/0.474 when using the segment and edge features. These features also outperform those of [152, 151] on all motion classes.

Table VIII.4 Comparison of feature sets on Peds2.

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Regression</th>
<th>MSE</th>
<th>err</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>right-slow</td>
<td>left-slow</td>
</tr>
<tr>
<td>area</td>
<td>GP-lin</td>
<td>0.8233</td>
<td>0.6237</td>
</tr>
<tr>
<td>segm</td>
<td>GP-lin</td>
<td>0.7953</td>
<td>0.5083</td>
</tr>
<tr>
<td>edge</td>
<td>GP-lin</td>
<td>0.8283</td>
<td>0.6063</td>
</tr>
<tr>
<td>segm+edge</td>
<td>GP-lin</td>
<td>0.6847</td>
<td>0.4743</td>
</tr>
<tr>
<td>[151]</td>
<td>GP-lin</td>
<td>0.7363</td>
<td>0.6140</td>
</tr>
<tr>
<td>[152]</td>
<td>GP-lin</td>
<td>0.7060</td>
<td>0.4913</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Regression</th>
<th>MSE</th>
<th>err</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>right-fast</td>
<td>left-fast</td>
</tr>
<tr>
<td>area</td>
<td>GP-lin</td>
<td>0.0130</td>
<td>0.0313</td>
</tr>
<tr>
<td>segm</td>
<td>GP-lin</td>
<td>0.0090</td>
<td>0.0037</td>
</tr>
<tr>
<td>edge</td>
<td>GP-lin</td>
<td>0.0233</td>
<td>0.0267</td>
</tr>
<tr>
<td>segm+edge</td>
<td>GP-lin</td>
<td>0.0097</td>
<td>0.0037</td>
</tr>
<tr>
<td>[151]</td>
<td>GP-lin</td>
<td>0.0170</td>
<td>0.0317</td>
</tr>
<tr>
<td>[152]</td>
<td>GP-lin</td>
<td>0.0203</td>
<td>0.0117</td>
</tr>
</tbody>
</table>

Figure VIII.15 shows the crowd count estimates (using the segment and edge features, and GP-lin) as a function of time, for the four motion classes. The estimates track the ground-truth well in most of the test frames, but misses one person in the “right-fast” class. The error is displayed in the 2nd image of the last row of Figure VIII.18, and shows that the cause is an error in the segmentation.
Figure VIII.9 shows the cumulative error for all four motion classes. On this scene, the system is within 1 person 93% of the time for the “right-slow”, and within 1 person 95% of the time for the “left-slow” class. On average, these estimates were within 0.485 and 0.415 of the ground-truth. Figure VIII.17 shows the average error for different crowd sizes. In this case, the error increases as the crowd size increases. For example, on the “right-slow” class, the average increases from 0.31 when 0 to 3 people are present, to 0.69 when the crowd contains 4 to 7 people, to 1.75 when the crowd contains 8 to 11 people. Finally, the feature set from [152] (Kong) performs similarly to the segment+edge features on the two slow classes, but does significantly worse on the two fast classes “right-fast” and “left-fast”. These results, again, suggest the efficacy of the system for performing robust crowd counting using low-level features. Figure VIII.18 shows the original image, segmentation, and crowd estimates for several frames in the test set.

Finally, we trained the system on 2000 frames of the dataset (every other frame), and ran the system on the remaining video. The resulting crowd estimates are shown in Figure VIII.19, while Figure VIII.20 shows several example outputs of the system (the video is also available from [52]). Similar to Peds1, the system tracks the changes in pedestrian traffic fairly well. Most errors tend to occur on objects that are not seen in the database, for example, three people pulling carts in first image of the fifth row of Figure VIII.20, or the small truck in the final image of Figure VIII.20. These errors are reasonable, considering that these objects were not seen in the training set, and the problem could be fixed by simply adding training examples of such cases.
Figure VIII.15 Crowd counting results on Peds2 over both the training and test sets for: (a) “right-slow”, (b) “left-slow”, (c) “right-fast”, (d) “left-fast. The gray bars show the two standard-deviations error bars of the GP.
Figure VIII.16 Cumulative error on Peds2 for: (a) “right-slow”, (b) “left-slow”, (c) “right-fast”, (d) “left-fast.

Figure VIII.17 Error splits for different crowd sizes on Peds1 for: (a) “right-slow”, (b) “left-slow”, (c) “right-fast”, (d) “left-fast.
Figure VIII.18 Crowd counting examples from Peds2: The red and green segments are the “right-slow” and “left-slow” crowds, and the blue and yellow segments are the “right-fast” and “left-fast” crowds. The estimated crowd count for each segment is in the top of each image, with the (standard-deviation of the GP) and the [ground-truth]. The ROI is also highlighted.
Figure VIII.19 Count estimates on 55 minutes of Peds2: (a) “right-slow” crowd; (b) "left-slow" crowd; (c) “right-fast” and “left-fast” crowds. The shaded bars indicate periods when the GP model had low confidence ($\sigma > 3$).
Figure VIII.20  Example counting results on 55 minutes of Peds2. The counts are in the top of each image, with the (standard-deviation).
VIII.E.4 Comparison with people detection algorithms

In this section, we compare the proposed crowd counting system with two state-of-the-art people detection algorithms. The first algorithm uses SVM and histogram-of gradients [155], which we denote “HOG”, while the second is based on a deformable parts model [156], which we denote “Deva”. The trained detection algorithms were provided by the respective authors. The algorithms were run on both databases, and a filter was applied to remove detections that were outside the ROI, inconsistent with the geometry of the scene, or had low confidence. To evaluate the detection algorithms, each detection was uniquely mapped to the closest ground-truth pedestrian, and a true positive (TP) was recorded if the ground-truth location was within its associated detection bounding box. A false positive (FP) was recorded otherwise. An ROC curve was then computed by varying the confidence threshold. Figure VIII.21 shows the ROC curves for HOG and Deva on the Peds1 and Peds2 databases. HOG outperforms Deva on both videos, with a smaller FP rate per image. However, neither algorithm is able to achieve a very high TP rate (the maximum TP is 74% on Peds1), due to the large amount of people occlusions in the video.

![Figure VIII.21 ROC curves for people detection algorithms on the (a) Peds1 and (b) Peds2 databases.](image)

Next, each algorithm was used to count the total number of people in the scene. For each algorithm, a confidence threshold was chosen that minimized the
count error on the training set. The count errors on the test set are shown in Table VIII.5, and the crowd counts are displayed in Figure VIII.22 and Figure VIII.23. For crowd counting, Deva has a lower average error rate than HOG (4.01 versus 5.32). This is an artifact of the high FP rate for Deva; the false detections artificially boost the count even though the algorithm has a lower TP rate. On the other hand, HOG always underestimates the crowd count, which is evident in Figure VIII.22 and Figure VIII.23.

Table VIII.5 Counting results using low-level features (GP) and people detection (HOG, Deva).

<table>
<thead>
<tr>
<th>Method</th>
<th>abs. err</th>
<th>bias</th>
<th>variance</th>
<th>abs. err</th>
<th>bias</th>
<th>variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP</td>
<td>1.787</td>
<td>1.186</td>
<td>3.317</td>
<td>0.7542</td>
<td>0.1959</td>
<td>0.8817</td>
</tr>
<tr>
<td>Deva [156]</td>
<td>4.012</td>
<td>1.621</td>
<td>22.1</td>
<td>1.565</td>
<td>−0.9833</td>
<td>3.68</td>
</tr>
<tr>
<td>HOG [155]</td>
<td>5.321</td>
<td>−5.315</td>
<td>11.51</td>
<td>2.607</td>
<td>−2.595</td>
<td>4.103</td>
</tr>
</tbody>
</table>

Finally, we compare with the proposed system, GP regression with low-level features. Since the detection algorithms only produce an overall count, we estimate the total GP count as the sum of the counts from the different motion classes. Both detection algorithms perform significantly worse than GP. In particular, the average test error of the detection algorithms is more than double that of the GP (e.g. 4.01 for Deva versus 1.79 for GP on Peds1). Figure VIII.24 shows the test error for different crowd sizes. The test error using the detection algorithms increases significantly as the crowd size increases. On the other hand, the test error using GP remains relatively constant over the various crowd sizes. These results demonstrate that the proposed counting system performs well above the state-of-the-art detection algorithms at the task of crowd counting.

VIII.F Summary and discussion

In this chapter, we have presented a vision system for estimating the size of inhomogeneous crowds, composed of pedestrians traveling in different directions,
Figure VIII.22 Counting results on Peds1 using: a) low-level features and GP; and people detection algorithms: b) HOG [155]; c) Deva [156].

Figure VIII.23 Counting results on Peds2 using: a) low-level features and GP-RR; and people detection algorithms: b) HOG [155]; c) Deva [156].

Figure VIII.24 Test error for different crowd sizes on the a) Peds1 and b) Peds2 databases.
that does not depend on object detection or feature tracking. The system was validated on two large datasets, and outperformed counting results using state-of-the-art people detection algorithms. The counting system was successfully applied to two hours of video, suggesting that the system could be used in a real-world environment for long periods of time.

Finally, we note that this type of crowd-counting system, which is based on extracting low-level features from crowd segments, would not be possible without the robust motion segmentation provided by the mixture of dynamic textures. Indeed, any errors that originate from the motion segmentation will propagate throughout the system, affecting the crowd counts. Hence, the efficacy of the system attests to the robustness of both the feature-regression methodology and the underlying mixture of dynamic textures segmentation algorithm.

### VIII.G Acknowledgements

The authors thank Jeffrey Cuenco for part of the ground-truth data, Navneet Dalal and Pedro Felzenszwalb for the detection algorithms from [155] and [156], and Piotr Dollar for running these algorithms.

The text of Chapter VIII, in part, is based on the material as it appears in: A. B. Chan, Z. S. J. Liang, and N. Vasconcelos. “Privacy preserving crowd monitoring: counting people without people models or tracking”, in *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), June 2008. The dissertation author was a primary researcher and an author of the cited material.
Chapter IX

Application: Modeling music with dynamic textures
IX.A Introduction

Throughout this thesis, we have demonstrated that the dynamic texture is a simple and robust model for video, with numerous applications in computer vision. It is important to note, however, that the dynamic texture models originally proposed for vision, are in fact generic probabilistic models, suitable for analyzing a variety of time-series data. In this chapter, we apply the dynamic texture model to the domain of computer audition.

It is common practice in music information retrieval to represent a song as a bag of audio feature-vectors (e.g., Mel-frequency cepstral coefficients). While this has shown promise in many applications, e.g. music annotation and retrieval [160], audio similarity [161] and song segmentation [162], the bag-of-features representation is fundamentally limited by its assumption that the feature-vectors are independent of each other, i.e., the representation ignores the dependencies between feature-vectors. As a result, the bag-of-features fails to represent the rhythmic qualities (e.g., tempo and beat patterns) and temporal structure (e.g. repeated riffs and arpeggios) of the audio signal. In this chapter, we consider simultaneously modeling both the spectral and rhythmical qualities of a music clip as a dynamic texture, a generative probabilistic model that models both the timbre of the sound, and its evolution over time. We apply this new audio representation to the task of automatic song segmentation.

The goal of automatic song segmentation is to automatically divide a song into self-coherent units such as the chorus, verse, bridge, etc. Foote [163] segments music based on self-similarity between timbre features. Goto adds high-level assumptions about repeated sections to build a system for automatically detecting choruses [164]. Turnbull et al. [165] present both an unsupervised (picking peaks of difference features) and supervised (boosted decision stump) method for identifying musical segment boundaries (but not labeling the segments themselves). Other methods attempt to explicitly model music and then cast segmentation as a
clustering problem. Gaussian mixture models (GMMs) ignore temporal relations between features but have worked well for segmentation and similarity [162] as well as classification of a variety of semantic musical attributes [160]. Hidden Markov models (HMMs) consider transitions between feature states and have offered improvements for segmentation [166] and genre classification [167]. Abdallah et al. [168] incorporate prior knowledge about segment duration into an HMM clustering model to address the problem of over-segmentation. Levy and Sandler [169] realize that feature-level HMMs do not encode sufficient temporal information and constrain their clustering based on the musical structure.

In contrast to these methods which do not explicitly model the temporal qualities of the signal, we introduce a new segmentation algorithm that accounts for both the rhythmic and timbral qualities of the signal. In particular, we cluster sequences of audio feature-vectors, extracted from the song, using the mixture of dynamic textures. The new algorithm explicitly models the temporal dynamics of the musical texture, capturing more of the information required to determine the structure of music.

IX.B Dynamic texture models for music

Although the dynamic texture (DT) and dynamic texture mixture (DTM) models were originally proposed in the computer vision literature as generative models for video sequences, they are generic models that can be applied to any time-series data. In this chapter, we will use the dynamic texture to model a sequence of audio feature vectors extracted from a song (e.g., a sequence of Mel-frequency cepstral coefficients). In particular, the DT encodes the sound component (audio feature at time $t$) with the observed variable $y_t \in \mathbb{R}^m$, and the dynamics (evolution of the sound over time) with the hidden state variable $x_t \in \mathbb{R}^n$. Because we are modeling sequences, we are able to capture both the instantaneous audio content (e.g., instrumentation and timbre), and the melodic and rhythmic content.
(e.g., guitar riffs, drum patterns, and tempo), with a single probabilistic model.

IX.B.1 Song segmentation

A song is segmented into sections (e.g. verse, chorus, and bridge) by extracting a set of sequences from a song using a sliding window, and clustering them with a dynamic texture mixture (DTM). The segmentation is performed using a coarse-to-fine approach. First, audio features-vectors are extracted from the audio signal (e.g., Mel-frequency cepstral coefficients). To produce a coarse segmentation, short sequences are extracted from the full sequence of audio feature-vectors using a sliding window (\(\sim 5 \text{ sec}\)) with a large step-size (\(\sim 0.5 \text{ sec}\)). A DTM is learned from the collection of windowed sequences using EM (Section IV.C), and the coarse song segmentation is formed by assigning each windowed sequence to the component with largest posterior probability. This first segmentation is relatively coarse (at best within 0.25 sec), due to the large step-size and the poor-localization properties of using a large window. Next, we refine the boundaries of the coarse segmentation. Sequences are extracted from the song using a smaller sliding window (\(\sim 1.75 \text{ sec}\)) and a finer step-size (\(\sim 0.05 \text{ sec}\)). A fine-grain segmentation is formed by assigning these sequences to the most-likely components of the DTM learned in the coarse-segmentation. Finally, the boundaries of the coarse segmentation are refined by searching for the closest boundaries in the fine-grain segmentation, producing the final segmentation. Note that using a large 5 second window for the coarse segmentation allows the DTM to model musical characteristics with long temporal durations (e.g., beat patterns, riffs, sustained notes, etc.). This is not possible when using a shorter window.

IX.C Experimental evaluation

In this section, we present experiments on song segmentation and boundary detection with the mixture of dynamic textures, along with an experiment on
song segment retrieval.

IX.C.1 Song database

We experiment on 100 pop songs from the RWC music database (specifically, RWCMDB- P-2001) [170] where each song has been segmented into coherent parts by a human listener [171]. The segments are accurate to 10ms and are labeled with great detail. For this work we group the labeled segments into 4 possible classes: “verse” (i.e., including verse A, verse B, etc.), “chorus”, “bridge” and “other” (“other” includes labels such as “intro”, “ending”, “pre-chorus”, etc. and is also used to model any silent parts of the song). This results in a “ground truth” segmentation of each song with 4 possible segments classes. On average, each song contains 11.1 segments.

IX.C.2 Audio features

The content of each 22050Hz-sampled, monaural waveform is represented using two types of music information features:

**Mel-frequency cepstral coefficients:** The Mel-frequency cepstral coefficients (MFCCs), developed for speech analysis [172], describe the timbre or spectral shape of a short time piece of audio and are a popular feature for a number of MIR tasks, including segmentation [163, 162, 165]. We compute the first 13 MFCCs for half-overlapping frames of 256 samples (one feature vector every ∼ 6 msec). In music information retrieval, it is common to augment the MFCC feature vector with its instantaneous first and second derivatives, in order to capture some information about the temporal evolution of the feature. When using the DT, this is unnecessary since the temporal evolution is modeled explicitly by the DT.

**Chroma:** Chroma features have also been successfully applied for song segmentation [164]. They represent the harmonic content of a short-time window of audio
by computing the spectral energy present at frequencies that correspond to each of the 12 notes in a standard chromatic scale. We compute a 12-dimensional chroma feature vector from three-quarter overlapping frames of 2048 samples (one feature vector every $\sim 23$ msec).

IX.C.3 Song segmentation

The songs in the RWC database were segmented into $K = 4$ segments using the DTM method described in Section IX.B.1 on the MFCC or Chroma features, which we denote DTM-MFCC and DTM-Chroma, respectively. For DTM-MFCC, we use a window size of 900 MFCC frames and a step-size of 100 frames, while for DTM-Chroma, we use a window size of 600 Chroma frames and a step-size of 20 frames. The dimension of the hidden state-space of the DTM was $n = 7$ for MFCC, and $n = 6$ for Chroma.

For comparison, we also segment the songs using a Gaussian mixture model (GMM) trained on the same feature data [162]. We learn a $K = 4$ component GMM for each song, and segment by assigning features to the most likely Gaussian component. Since segmentation decisions are now made at the short time-scale of individual features, we smooth the GMM segmentation with a length-1000 maximum-vote filter. We compare the models against two baselines: “constant” assigns all windows to a single segment, “random” selects segment labels for each window at random.

We quantitatively measure the correctness of a segmentations by comparing with the ground-truth using two metrics: 1) the error rate, which is the proportion of the entire song that is assigned to an incorrect segment; 2), the Rand index [82], a clustering metric that intuitively corresponds to the probability that any pair of items will be clustered correctly, with respect to each other (i.e., in the same cluster, or in different clusters). We also report the average number of segments per song. The results are averaged over 100 songs.

Table IX.1 and Table IX.2 report the segmentation results for the MFCC
and Chroma features, respectively. DTM-MFCC outperforms all other models, with an error rate of 0.20 and Rand index of 0.79. GMM performs significantly worse than DTM, e.g., the error rate increases to 0.42 on the MFCC features. Both models tend to over-segment songs although this problem is less severe for DTM. This suggests that there is indeed a benefit in modeling the temporal dynamics with the DTM.

### IX.1 Song segmentation using MFCC features.

<table>
<thead>
<tr>
<th>Model</th>
<th>Error Rate</th>
<th>Rand</th>
<th># Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTM-MFCC</td>
<td>0.20</td>
<td>0.79</td>
<td>16.9</td>
</tr>
<tr>
<td>GMM-MFCC</td>
<td>0.42</td>
<td>0.66</td>
<td>58.7</td>
</tr>
<tr>
<td>Constant</td>
<td>0.59</td>
<td>0.35</td>
<td>1</td>
</tr>
<tr>
<td>Random</td>
<td>0.64</td>
<td>0.54</td>
<td>279.0</td>
</tr>
<tr>
<td>Truth</td>
<td>0.00</td>
<td>1.00</td>
<td>11.1</td>
</tr>
</tbody>
</table>

### IX.2 Song segmentation using Chroma features.

<table>
<thead>
<tr>
<th>Model</th>
<th>Error Rate</th>
<th>Rand</th>
<th># Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTM-Chroma</td>
<td>0.26</td>
<td>0.76</td>
<td>13.5</td>
</tr>
<tr>
<td>GMM-Chroma</td>
<td>0.46</td>
<td>0.60</td>
<td>24.1</td>
</tr>
<tr>
<td>Constant</td>
<td>0.58</td>
<td>0.32</td>
<td>1</td>
</tr>
<tr>
<td>Random</td>
<td>0.67</td>
<td>0.56</td>
<td>329.3</td>
</tr>
<tr>
<td>Truth</td>
<td>0.00</td>
<td>1.00</td>
<td>11.1</td>
</tr>
</tbody>
</table>

An example of the DTM segmentation of one song is compared to the ground truth in Figure IX.1 where we see that, while most of the DTM segments are accurate, there are some errors due to imprecise borders, and some cases where the model over-segments. More example segmentations are available in the supplemental [52].

### IX.C.4 Boundary detection

In addition to evaluating the segmentation performance of the DTM model, we can consider its accuracy in simply detecting the boundaries between
Figure IX.1 Example of the true segmentation of a song compared to the automatic GMM (top) and DTM (bottom) segmentations.

segments (without trying to label the segment classes). The song boundaries are computed by segmenting the song using DTM-MFCC with $K = 5$, and then finding the time instances where the segmentation changes. We compare results with Turnbull et. al [165], which tackles the boundary detection problem, using the same RWC data set, by learning a supervised classifier that is optimized for boundary detection. We also compare with the music analysis company EchoNest [173], which offers an online service for automatically detecting music boundaries.

The evaluation criteria are two median time metrics: true-to-guess and guess-to-true, respectively measure the median time from each true boundary to the closest estimate, and the median time from each estimate to the closest true boundary. The results are averaged over 100 songs and are presented in Table IX.3.

DTM-MFCC achieves both lower guess-to-true and true-to-guess times, indicating that DTM-MFCC is more accurate at finding the song boundaries. Note that DTM-MFCC is an unsupervised method, whereas the next best performer [165] is a supervised algorithm.
Table IX.3 Boundary detection using MFCC features.

<table>
<thead>
<tr>
<th>Model</th>
<th>Guess-to-True (sec)</th>
<th>True-to-Guess (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTM-MFCC</td>
<td>4.06</td>
<td>1.76</td>
</tr>
<tr>
<td>Turnbull et. al [165]</td>
<td>4.29</td>
<td>1.82</td>
</tr>
<tr>
<td>EchoNest [173]</td>
<td>5.09</td>
<td>1.84</td>
</tr>
</tbody>
</table>

IX.C.5 Song segment retrieval

Given the automatic segmentation of a song, we can retrieve other similar song clips in the database, answering questions like “what song sounds similar to the verse of this song?” We represent each song segment by its corresponding dynamic texture component in the DTM-MFCC, and measure similarities between dynamic textures with the Kullback-Leibler (KL) divergence (Chapter III). The five closest segments were retrieved for each song segment, and the results are available in the supplemental material [52]. Qualitatively, the system finds segments that are similar in both audio texture and temporal characteristics. For example, a segment with slow piano will retrieve other slow piano songs, whereas a rock song with piano will retrieve more upbeat segments. This indicates the dynamic texture model is capturing both the “texture” of the audio content (e.g., timbre and instrumentation), along with temporal characteristics (e.g. tempo, beat structures, style).

In order to visualize the distribution of songs in the database, the song segments were embedded into a 3-d manifold using local-linear embedding (LLE) [174] and the KL similarity matrix computed above. Two dimensions of the embedding are shown in Figure IX.2. We observed that these two axes of the embedding correspond to the tempo and beat of the segment (e.g., dance beat, hip-hop, rock, or mellow), and the instrumentation of the segment (e.g., piano, synthesizers, or distorted guitar). Again, this demonstrates that the dynamic texture model is successfully modeling both the audio texture and the temporal characteristics of the songs. Finally, we selected seven songs that are stylistically different, and highlight
them in Figure IX.2. While most songs are concentrated in specific regions of the manifold (e.g. the sections of the hip-hop song are similar sounding), some songs span multiple regions (e.g., the song highlighted in red contains piano in the verse, and fast upbeat rock in the chorus and bridge).

Figure IX.2  2-D visualization of the distribution of song segments. Each black dot is a song segment. Seven songs are highlighted in different colors, with segments marked as ◎ (verse), □ (chorus), ◇ (bridge), and △ (“other”).

IX.D Summary and discussion

In this chapter, we have applied the dynamic texture mixture model to the analysis of music time series. By describing music as a mixture of coherent textures, we demonstrate that the DTM model can accurately segment music and detects boundaries between segments as accurately as leading research and commercial systems. Examining a low-dimensional representation of the DTM-derived similarity between musical segments illustrates that the model is capturing both timbral and dynamical elements of the music and it shows promise as a new tool for automatic music analysis.
IX.E Acknowledgements

The text of Chapter IX, in full, is based on the material as it appears in: A. B. Chan, L. Barrington, and G. R. G. Lanckriet, “Dynamic texture models of music”, submitted for publication to International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2009. The dissertation author was a primary researcher and joint first author of the cited material.
Chapter X

Conclusions
In this thesis we have developed a family of dynamical models for video that expands on the modeling power of the dynamic texture. We have introduced two multi-modal generative models, the mixture of dynamic textures, which represents a collection of video sequences as samples from a set of dynamic textures, and the layered dynamic texture, which represents a video as a layered collection of dynamic textures of different appearance and dynamics. To enable modeling of complex motions, we proposed the kernel dynamic texture, which can represent motion along a non-linear manifold. We have also derived a probabilistic kernel, based on the Kullback-Leibler divergence between dynamic textures, which combines with large-margin classifiers to form a new framework for classifying dynamic textures. These new models provide a principled probabilistic framework for the analysis of video containing a wide variety of visual processes.

From a practical standpoint, we have successfully applied the proposed models to a wide variety of challenging computer vision problems, including motion segmentation, video classification, highway traffic monitoring, crowd counting, and background subtraction. Through extensive experimentation, we have demonstrated that the solutions based on dynamic texture models are more robust than those using traditional computer vision representations. This is, in significant part, due to the fact that the underlying generative probabilistic framework is capable of abstracting a wide variety of complex motion patterns into a simple spatio-temporal process. These successes validate the dynamic texture framework as a principled approach for representing video, and suggest that the models could be useful in other domains, such as computer audition, that require the analysis of time-series data.
Bibliography


