Duration Dependent Hidden Markov Models and Change Point Geometry for Video Surveillance

by

Brandon A. Mayer

B. S., Electrical Engineering, University Of Miami, 2008
B. S., Music Engineering and Technology, University Of Miami, 2008
Sc.M., Brown University, 2010

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in the
School of Engineering at Brown University

Providence, Rhode Island

May 2015
© Copyright 2015 by Brandon A. Mayer
This dissertation by Brandon A. Mayer is accepted in its present form by
the School of Engineering as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date ________________

Joseph L. Mundy, Ph.D., Advisor
School of Engineering

Recommended to the Graduate Council

Date ________________

Gabriel Taubin, Ph.D., Reader

Date ________________

Pedro Felzenszwalb, Ph.D., Reader

Approved by the Graduate Council

Date ________________

Peter M. Weber
Dean of the Graduate School
Curriculum Vitae

Brandon A. Mayer was born July 12, 1986 in Miami Florida U.S.A. He graduated *cum laude* from the University of Miami in 2008, receiving Bachelor of Science degrees in Electrical Engineering and Music Engineering and Technology. Brandon received the Tzay Y. Young award for "Best Senior Design Project" from the school of engineering at the University of Miami for his work on coupling active contours and conditional random fields for 3D medical image segmentation. Brandon completed the Masters of Science degree in 2010 at Brown University where he continued publishing contributions in the fields of Machine Learning, Computer Vision, Computational Game Theory and Climate Informatics while developing Change Detection algorithms for video surveillance with the guidance of Dr. Joseph L. Mundy.
Acknowledgements

This thesis is dedicated to Michele, Stephen, Claire, and Harley Mayer.
Contents

List of Tables ix

List of Figures xii

1 Introduction 1

1.1 Contributions and Outline ............................................. 10

2 Pixel-level change detection 12

2.1 Preliminaries ................................................................. 13

2.1.1 Notation ................................................................. 13

2.1.2 Generative Probabilistic Models .................................... 14

2.1.3 Probabilistic Graphical Models .................................... 15

2.2 Gaussian mixture models .................................................. 16

2.3 Model Selection and Bayesian Methods for Gaussian mixture models ... 21

2.4 Hidden Markov Models .................................................... 26

2.5 Nonparametric Pixel Intensity Models ................................ 29

2.5.1 Kernel Density Estimation ........................................... 30

2.5.2 Sample Distance Methods .......................................... 34

2.6 Codebook based Pixel Intensity Models ............................... 38

2.7 Domain Transformation Pixel Models ................................ 44
3 Modeling intensity sequences for pixel-level change detection

3.1 Per-Pixel Intensity Sequence Model ................................................. 53
3.2 Adaptive Duration Distribution & Epoch Granularity ...................... 57
3.3 Learning DDHMM parameters .......................................................... 62
3.4 Qualitative Sample Analysis ............................................................. 67
3.5 Empirical DDHMM Complexity ......................................................... 75
3.6 DDHMM Evaluation: Likelihood ......................................................... 80
  3.6.1 Forward program to compute DDHMM log-likelihood ................. 81
  3.6.2 Results ...................................................................................... 83
3.7 Duration Dependent Codebook Classifier ......................................... 84
3.8 Change Localization by State Merging and Splicing .......................... 84
3.9 Change Detection Results ................................................................. 85

4 Region based change detection via change point geometry ............... 92

4.1 Related Region-Level Algorithms ..................................................... 93
  4.1.1 Spatio-Temporal Local Binary Patterns (STLBP) ....................... 93
  4.1.2 Dual Model Gaussian Mixtures (SGMM-SOD) ............................ 99
4.2 Change Point Geometry .................................................................. 102
4.3 Constructing Change Hulls via Efficient Graph Segmentation .......... 109
  4.3.1 Disjoint-Set Forests ................................................................. 120
  4.3.2 Performance Analysis of Change Point Hull Construction .......... 122
4.4 Change Point Hull Eigen-Shape ......................................................... 140
4.5 Coupling Duration Dependent Hidden Markov Models and Change Point Geometry for Video Surveillance .................................................. 143
  4.5.1 Computing the Direction of Moving Objects .............................. 143
  4.5.2 Detecting Change ....................................................................... 145
List of Tables

2.1 Abbreviations and summary of reviewed change detection methods. . . . . . . . . . . 13
2.2 Summary of parameters for algorithms based on Gaussian Mixture Models as de-
scribed by [1] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.3 List of parameters for PBAS with summary and parameter values suggested by [2]
and used in all experiments. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
2.4 Enumerate parameters of ViBe with empirically determined default values suggested
by [3] which were used in all experiments. The first four parameters were documented
in the original ViBe paper [4], the remaining were added as improvements in [3]. . . 38
2.5 Performance comparison of processing and memory requirements for CBBGS, SG [1]
and GKDE [5] as reported in [6]. Note that for a given parameter setting, the memory
requirements of the SG and GKDE algorithms are fixed. However, due to the adaptive
codebook construction, the CBBGS memory requirements will change depending on
observed scene dynamics. The value reported for the storage requirement of CBBGS
is the computed with respect to the average codebook size which for the test scene
was reported to be four codewords. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
2.6 Table of fundamental parameters for the CBBGS algorithm [6]. . . . . . . . . . . . 43
2.7 Parameters of Wave-Backand experimental values reported in [7]. . . . . . . . . . 48
2.8 Waviz parameters and experimental values reported in [8]. In all experiments \( K = N \) and experiments were conducted with \( N = 32, 64, 128 \) where it was determined empirically that \( N = K = 32 \) gave the best results.

3.1 Summary of notation used for Duration Dependent Markov Model.

3.2 DDHMM learning algorithm parameters and default values. If a parameter is labeled “Global”, then the single parameter applies to models at each independent pixel location.

3.3 Average number of DDHMM states created, rounded to the nearest integer, for each video sequence.

3.4 Log-likelihood of the test observation sequences according to each model. Values closer to zero are desirable. Note that these measurements are on the log scale and therefore represent an increase in likelihood by a factor of 2-3 when comparing the HMM and DDHMM.

3.5 Fields Point: Quantitative change detection results showing True Positive Rate (TPR), Precision, False Positive Rate (FPR) and False Negative Rate (FNR).

3.6 Swing: Quantitative change detection results showing True Positive Rate (TPR), Precision, False Positive Rate (FPR) and False Negative Rate (FNR).

4.1 STLBP algorithm parameter summary.

4.2 SGMM-SOD algorithm notation and parameter summary.

4.3 Key variables for the per-pixel finite state machine classifier. \( I \) represents the short- and long-term per-pixel classification, \( \{F_L, F_S\} \), at a given pixel. \( Q \) are the semantic interpretations of the dual model detections: Background, Moving Object, New Static Foreground, and Uncovered Background and \( Z \) enumerates the states of \( Q \) [9].

4.4 Summary of notation of summary used for constructing change point hulls.
4.5 Run-time performance for building change point hulls. The resolution of each video, the number of change point vertices, number of edges, number of resulting change point hulls after segmentation and the run time in milliseconds of the graph segmentation algorithm is shown for each experiment.

4.6 Summary of proposed region-level change detection parameters.
List of Figures

1.1 CCTV digital video surveillance system, military-grade Unmaned Aerial Vehicle (UAV) and a commercial drone equipped with high definition video cameras. ............ 2

1.2 As one of the first western countries to adopt state-sponsored CCTV video surveillance systems, many studies concerning the prevalence and effectiveness of CCTV systems originate from the UK. Figure 1.2a shows estimates of the number of CCTV surveillance cameras per 1,000 people in London boroughs while figure 1.2b visualizes the number of cameras per 1,000 people across the UK. These statistics were compiled in 2009 with graphics taken from [10]. ......................................................... 3

1.3 Video surveillance footage of Boston Marathon 2013 bombing suspects. Figure 1.3a shows suspects maneuvering the crowd to place bombs in the form of pressure cookers contained in their backpacks. Figure 1.3b is a frame of unreleased video footage compiled by the FBI showing one of the suspects, leaving a backpack in a crowded area before fleeing the scene. Images were taken from [11] and [12] respectively. ... 4

1.4 Appearance of real-world scenes vary markedly over time with predictable epochs of recurring change. Image taken from [13]. .......................... 6

1.5 Detecting vehicles in video surveillance. Figure 1.5a originally appeared in [14] and figures 1.5b and 1.5c are adopted from [15]. .......................... 9
2.1 Example of a probabilistic graphical model. This directed graphical model specifies the following conditional factorization of the joint probability distribution over random variables $A$ and $B$: $p(A, B) = p(A) p(B | A)$.  

2.2 Specifying an independent factorization over a collection of observations which are conditionally dependent on a random variable. Random variables are shown as clear nodes with dotted outlines while observations are represented as shaded circles. Figure 2.2b is an equivalent, condensed representation of figure 2.2b using plate notation. 

2.3 Visualizing a Gaussian mixture model using Probabilistic Graphical Models. Shaded nodes represent observed quantities and circles with dotted outlines are random variables. 

2.4 Visualization of DP-GMM using probabilistic graphical model plate notation adapted from [16]. The $\alpha$ and $H$ random variables encapsulate the Dirichlet process prior while the $\mu$ and $\Lambda = \{\nu_k, \kappa_k, \sigma_k\}$ nodes represent the component dependent variables, taken to be random quantities within the Bayesian Framework. Latent component indicator variables $S_t$ are used to independently model pixel observations $y_t$, $t \in \{1, \ldots, T\}$. The process is replicated independently for each pixel in a frame of video $F$. 

2.5 Geometric distributions computed according to equation 2.19 for various values of $A(k, k)$. 

xiii
2.6 Visualizing Gaussian kernel density estimation. The distribution to be estimated is a Mixture of Gaussians with two components and parameters \( \theta = \{ \omega_0 = 0.4, \mu_0 = 25, \sigma_0 = 20, \omega_1 = 0.6, \mu_1 = 200, \sigma_1 = 10 \} \). The probability density of the true mixture distribution is visualized as the solid green line in each plot. Samples drawn from the true distribution are shown as red dots along the \( y = 0 \) axis. Gaussian kernels centered at each sample are shown as dotted black lines and the corresponding density estimate, computed according to equation 2.21 is shown as the magenta dotted line. To show the effect of kernel bandwidth, each row shows the estimated density for a different value of \( \sigma \). The number of samples used in the estimation increases from left to right to show how more samples lead to more accurate results for a given kernel bandwidth.

2.7 Fixed camera video surveillance footage taken from at the entrance of a parking lot. Pixel \((240, 180)\) \((x, y)\) frame coordinates) is highlighted by the red circle.

2.8 Per-pixel Gaussian Kernel Density estimate proposed by Elgammal et. al. [5] for the pixel location highlighted in figure 2.7.

2.9 Visualizing the estimated values of the decision threshold \( R \) as computed by PBAS where larger values of \( R \) are visualized are more white and smaller black in the left frame. The leaves blowing in the wind shown in the right figure yield higher variance in the minimal distance, inducing large values of \( R \) compared to the pixels associated with the street or static objects in the scene. Image taken from [2].

2.10 Codeword color space under illumination changes. The color board shown in figure 2.10a was imaged under various illumination conditions and the resulting color observations are shown in RGB colorspace in figure 2.10b. Figure originally appears in [6].
2.11 CBBGS codeword geometry. A per-pixel codeword defines a cylindrical decision boundary in the RGB color space. For an observation $y_t$ to be considered a match to a particular codeword $c_m$, the observation must lie inside the codeword’s cylindrical boundary. The color distance threshold $\epsilon$ constrains the radius of the boundary while the brightness constraints controls the height. Matching observations adapt the position and orientation of the decision boundary. Image adapted from [6].  

2.12 Marginalizing temporal information of distinct signals may lead to indistinguishable indistinguishable static distributions. Figure originally appeared in [7].

2.13 Distance trail and removal adapted from [7].

3.1 Visualizing the general HSMM as a probabilistic graphical model. A sequence of observations $Y = \{y_1, \ldots, y_T\}$ of length $T$ is broken up into $L$ segments or sub-sequences that are related to the latent state-duration pairs through a conditional emission density. Note, $L$ is a random variable but is omitted from the illustration for clarity.

3.2 A concrete example of an HSMM. The duration random variable at each segment is conditioned on the duration of the previous segment and the current state. The state transition distribution, in this example, is dependent on the previous state and the previous duration. The likelihood of each observation sub-sequence is dependent on the duration random variable but the links are omitted for clarity.

3.3 Visualizing the temporal independence assumptions of Duration Dependent Markov Model. The duration distribution is conditioned only on the current state and the observations sequence is modeled as a collection of independent identically distributed (i.i.d.) random variables with size equal to the value of the segment duration. Plate notation is used to concisely represent the i.i.d. emission density assumption given a state-duration pair.
3.4 Alternative DDHMM representation using counter and finishing random variables.

To establish a graphical model for the DDHMM with a fixed topology, rather than directly modeling duration, counter and finishing random variables are introduced. There is a state indicator random variable \( \{S_1, \ldots, S_T\} \) for each observation, \( \{y_1, \ldots, y_T\} \).

The counter random variables \( \{C_1, \ldots, C_T\} \) represent the remaining time left in given homogeneous state segment while the finishing random variables \( \{F_1, \ldots, F_T\} \) takes the value of one only when there is a state transition \( S_t \neq S_{t-1} \) and zero elsewhere.

This figure was influenced by similar graphical models appearing in [17, 18] and [19].

3.5 Visualizing the theoretically absolute time line in terms of relative time between events. Each state conditional duration distribution of the DDHMM measures the relative time between two events, when a particular state \( S_t = s_k \) begins, and when the system transitions to any other state, \( S_{t+1} \) with \( S_{t+1} \neq s_k \). At the correct time scale, only a single event can happen per temporal quantum.

3.6 A forest of binary trees. An ordered collection of trees are maintained to adaptively quantize the time line. The forest root shown as the green dot and component binary trees shown in blue.

3.7 Partitioning a segment of the time line with a single binary tree component. (a) shows the relationship between the depth parameters \( \Delta_{\min} \) and \( \Delta_{\max} \) with corresponding temporal segments in (b).

3.8 Numerical example of the proposed adaptive histogram with \( \Delta_{\min} = 4, \Delta_{\max} = 0 \) after observing the sequence \( \{19, 18, 17, 18\} \). The first value, 19 creates three trees to span the range 1-24, the values 18,17,18 further refine the third binary tree resulting in the quantization of the time line by the red bins with adaptive widths.
3.9 Swing video posterior sampling evaluation. The intensity sequence observed at the pixel location highlighted in figures 3.9a, 3.9b, and 3.9c is shown in figure 3.9d. Figures 3.9e and 3.9f show intensity sequence samples drawn from GMMs and HMMs respectively, trained using the Expectation Maximization algorithm for various numbers of states $K$. Figure 3.9g shows sample intensity sequences drawn from a DDHMM trained using the learning algorithm proposed in section 3.3.

3.10 Highway posterior sampling evaluation. The intensity sequence extracted from the Highway video sequence at the pixel location at the center of the circle highlighted in figures 3.10a, 3.10b, and 3.10c is shown in figure 3.10d. The observed intensity sequence was used to train GMMs and HMMs for various number of states, $K$, using the Expectation Maximization algorithm. Figures 3.10e and 3.10f show intensity sequences sampled from the learned GMM and HMMs respectively. These samples are compared to the intensity sequences sampled from the DDHMM model and learning algorithm proposed in section 3.3 which are shown in figure 3.10g.

3.11 Fields Point posterior sampling evaluation. The intensity sequence extracted from the Fields Point video sequence at the pixel location at the center of the circle highlighted in figures 3.11a, 3.11b, and 3.11c is shown in figure 3.11d. The extracted intensity sequence was used to train GMMs and HMMs for various number of states, $K$, using the Expectation Maximization algorithm. Figures 3.11e and 3.11f show intensity sequences sampled from the learned GMM and HMMs respectively. These samples are compared to the intensity sequences sampled from the DDHMM model and learning algorithm proposed in section 3.3 and shown in figure 3.11g.
3.12 Tehachapi posterior sampling evaluation. The intensity sequence observed in the Tehachapi video sequence at the pixel location at the center of the circle in figures 3.12a, 3.12b and 3.12c is shown in figure 3.12d. Intensity sequences sampled from GMMs and HMMs with various number of states, trained using the observed sequence and the Expectation Maximization algorithm are shown in figures 3.12e and 3.12f. Samples of intensity sequences generated by a DDHMM trained according to the algorithms in section 3.3 are shown in figure 3.12g for comparison.

3.13 Helix posterior sampling evaluation. The intensity sequence observed in the Helix video sequence at the pixel location at the center of the circle in figures 3.13a, 3.13b, and 3.13c is shown in figure 3.13d. Intensity sequences sampled from GMMs and HMMs with various numbers of states, $K$, trained using the Expectation Maximization algorithm are shown in figures 3.13e and 3.13f. Samples of intensity sequences generated by a DDHMM trained according to the algorithms developed in section 3.3 are shown in figure 3.13g for comparison.

3.14 Entropy of empirical intensity distribution and number of states at each per-pixel DDHMM for the Fields Point video sequence.

3.15 Entropy of empirical intensity distribution and number of states at each per-pixel DDHMM for the Helix video sequence.

3.16 Entropy of empirical intensity distribution and number of states at each per-pixel DDHMM for the Tehachapi video sequence.

3.17 Empirical intensity distribution computed within the highlighted area over all frames of the Tehachapi video.

3.18 Entropy of empirical intensity distribution at each pixel and number of states at each per-pixel DDHMM for the Swing video sequence.

3.19 Entropy of empirical per-pixel intensity distribution and number of states at each DDHMM for the Highway video sequence.
3.20 Empirical intensity distribution computed over all frames at pixel locations within the highlighted section. ................................................................. 80

3.21 Visualizing Merge and Splice Hypotheses: The localization module does not modify the underlying state assignment but merely suppresses false positive artifacts arising from changes in the scene disrupting the duration of otherwise normal states and helps localize the change point in time. .................................................. 85

3.22 Fields Point: Frames of video of the Fields Point sequence are shown in the first row with time flowing from left to right across the columns. The second row shows hand-labeled ground truth masks corresponding to the frame in the same column. The third through ninth rows show the change detection results of the proposed, SG-GMM, CBBGS, STLBP, PBAS, ViBe, STLBP and SGMM-SOD algorithms for the example frame in the corresponding row. The proposed method is the only algorithm able to learn that the spinning blades are a normal part of the scene and can still detect the previously unobserved cars. The results of the SG-GMM algorithm were obtained using the implementation provided by [20]. ........................................ 88

3.23 Swing sequence: Change detection results for algorithms run on the Swing video are shown as binary masks. A white pixel indicates change was detected at the location whereas black is no change. Each row shows the output of a different change detection algorithm for the example frames in figure 3.23a where time flows from left to right. Hand-segmented groundtruth is shown in figure 3.23b and the output of the proposed DDHMM codebook, SG-GMM, CBBGS, PBAS, ViBe, STLBP, SGMM-SOD algorithms are shown in the remaining rows. The proposed method is the only algorithm that learns the swinging child is a normal part of the scene but still detects the previously unobserved pedestrian. The results of the SG-GMM algorithm were obtained using the implementation provided by [20]. ........................................ 90
4.1 Example computing the LBP descriptor at the center pixel using an eight pixel neighborhood where each pixel is labeled zero through seven. First, the intensity values at each neighbor is compared to the intensity value at the center pixel and each neighbor is set to the value of one if its intensity is greater than the intensity at the center pixel or zero otherwise. The results are then multiplied by two raised to the power of the neighbor index. Finally, the results from the previous step are summed over all neighbors to assign a single LBP code to the target pixel. In this example the final LBP code for the center pixel (shown in gray) is 143. This example originally appeared in [21].

4.2 Results from [22]. Top row are example frames from a video sequence. The second row is hand-labeled ground truth where a white pixel denotes a location corresponding to significant change in the video frame and black is no change. The third row visualizes the change detection classification from the STLBP algorithm. The fourth is the output of the SG-GMM algorithm and the fifth is the GKDE algorithm. Image taken from [22].

4.3 SGMM-SOD region classification example. Walking pedestrians are detected by both the long and short term background models and are displayed in green. The luggage initially left behind by the pedestrian in the middle frame is absorbed into the short term model but detected as change by the long term model. The bounding box is well-formed and therefore classified as a new static object visualized in yellow. When the luggage is removed, the edges of the change region match the long-term background model and the change is classified as a removed object shown in gray. Images are from [9].
4.4 Visualization of proposed algorithm. Each pixel of an input video is segmented into intensity subsequences with similar appearance and duration using a Hidden Semi-Markov Model. The beginning of each subsequence, a 3D point in 2D space-time and known as a change point vertex, are clustered into groups with similar appearance using a graph segmentation algorithm to form a change point hull. The vertices within a change point hull are used to construct a spatial covariance matrix which is decomposed via Singular Value Decomposition (SVD) to obtain the eigenvector and eigenvalues of the change point hull. The eigenvectors and eigenvalues are then used as descriptors for coherent motion and detecting meaningful scene aberration.

4.5 Crossing-Paths video. Figures 4.5a through 4.5e are frames from the Crossing-Paths video. Figures 4.5f, 4.5g and 4.5h are a front, top and rear view of the change point vertices in 3-d image space-time. Vertices with matching colors belong to the same change point hull.

4.6 Figure 4.6a shows the independent per-pixel temporal segmentation of intensity subsequences into homogeneous groups with similar appearance and duration. The beginning of each sequence, known change point vertices, are shown in fig. 4.6b as black spherical points in 3D (2D image space and time). Each change point vertex is connected to its four nearest neighbors and assigned an edge weight proportional to appearance and duration similarity measures and is used for segmenting the change hull.
4.7 Crossing paths 1-d example. A pair of 1-dimensional objects traveling across a scene in opposite directions at constant speed are imaged by a one-dimensional camera, visualized by as the red line. The path of change point vertices associated with the object moving from left to right is visualized with green dots. The path of the change point vertices associated with the object moving in the opposite direction is shown as a dotted blue line. The vertices of each object trace a line in 2-d space-time where the magnitude and sign of the line’s slope is proportional to object velocity and direction.

4.8 Visualizing the change point geometry induced by the periodic motion of the wind turbine in the Fields Point video sequence. The rotating turbine blades induce change point vertices with a global helix-like structure in 3D space-time.

4.9 Examples of a graph without cycles and a graph with a cycle.

4.10 Simple example of computing the minimum spanning tree for a weighted undirected graph. Figure 4.10a shows a particular graph for which the minimum spanning tree is to be computed. There are three possible spanning trees for the graph, \( T_1, T_2, \) and \( T_3, \) shown in the figures 4.10b, 4.10c and 4.10d respectively. The sum of the edge weights for spanning tree \( T_1 \) is eleven, the sum of edge weights for \( T_2 \) is fifteen and the sum of edge weights for \( T_3 \) is twelve. Therefore, \( T_1 \) is the minimum spanning tree.

4.11 Computing the minimum spanning tree of the example graph in figure 4.10. The list of edges sorted by weight is \( \{e_3, e_5, e_2, e_4, e_5\} \). Kruskal’s algorithm iterates over the sorted edge list iteratively growing the minimum spanning tree. The first four iterations of Kruskal’s algorithm are visualized in figures 4.11a, 4.11b, 4.11c and 4.11d where the highlighted edges represent the members of \( T^{\text{min}} \) as it is updated in each step. The solution \( T^{\text{min}} = T_1 \) is the same as when computed by brute force enumeration by inspection in figure 4.10.
4.12 Visualizing disjoint set forests. Component sets are represented as disjoint trees and a segmentation is the set of all disjoint trees. Elements of the tree point to their parents and the root of the tree points to itself. In this example there are three component sets identified by their roots, $C_1 = a$, $C_2 = e$, $C_3 = f$. $C_1$ contains the elements $\{a, b, c, d\}$, $C_2$ contains the single element $\{e\}$, and component $C_3$ contains $\{f, g, h\}$.

4.13 Run-time performance building change point hulls using the graph segmentation algorithm of Felzenszwalb and Huttenlocher [23]. As seen in the figure, the algorithm for grouping change point vertices scales linearly with the number of edges between change point vertices as predicted by the theoretical analysis.

4.14 Black-Square video sequence. Figures 4.20a, 4.20b, 4.20c and 4.20d show example frames taken from the Black-Square video sequence which is a synthetically generated scene consisting of a black square traversing a white background. Figures 4.14e and 4.14f show a front and top view of the change vertices and hulls respectively. Figure 4.14g shows a straight on view of the change vertices and hulls of the moving object after removing the initial coplanar vertices lying on the spatio-temporal positions of the first frame. By inspection of the change point vertices and hulls, it may be seen that an object traversing the camera plane with constant velocity will induce a collection of coplanar change point vertices at an angle with respect to the camera plane.
4.15 *Black-Square-Occlusion* video sequence. Figures 4.15a through 4.15d show example frames taken from the Black-Square-Occlusion sequence which consists of a black square moving from left to right across a white background. The moving black square is occluded at the frame center by a gray rectangle. Figures 4.15e, 4.15f, and 4.15g visualize the resulting change point vertices and hulls in 3-d image space-time. The change point geometry consists of two disjoint sets of coplanar change point vertices lying at an angle with respect to the camera frame. These sets of coplanar points correspond with the change vertices created to account for the moving black square.

4.16 *Synthetic-Car-Down* video sequence. This is a synthetic scene of a black square traveling down a gray strip surrounded on both sides by white pixels. Figures 4.16a through 4.16c are example frames taken from the original video. Figures 4.16e and 4.16f visualize the change point vertices and hulls in 3-d image space-time that are created to account for the scene’s appearance dynamics.

4.17 *Synthetic-Car-Down-Noise* video sequence. This video sequence is the same as in figure 4.16 except uniform noise on the interval \([-20, 20]\] has been added to every gray value as seen in the example frames in figures 4.17a, 4.17b, 4.17c, and 4.17d. The DDHMM is able to account for the additional pixel-level intensity variance but there are some mistakes. However, these mistakes will be accounted for in a later section when change point hulls are filtered by size (number of member vertices) and shape.

4.18 *Synth-Spin* video sequence. Figures 4.18a through 4.18h show example frames of the Synth-Spin video sequence consisting of a black line rotating counter-clockwise against a white background. The resulting change point geometry shown in figures 4.18i and 4.18j from two vantage points and mirrors a spiral staircase or helix in 3-d image space-time.
4.19 *Walk-Forward* video sequence and change point hull segmentation. A single pedestrian walks straight through a scene at constant speed as seen in the example frames of figures 4.19a, 4.19b and 4.19c. At the beginning of the sequence, the DDHMM algorithm creates an initial states for the color of the wall background. When the pedestrian enters the scene, additional state-duration pairs are created to account for the pedestrians appearance and motion dynamics. The change point vertices are shown in figure 4.19d. Figures 4.19f and 4.19g show the change point hulls in 3D image space-time from a side and top view respectively where color denotes common hull membership and black is empty space. Figure 4.19e shows a straight on view of the change point vertices and hulls associated with the pedestrian over time.

4.20 *Stop-Forward* video sequence. Figures 4.20a through 4.20d show example frames taken from the Stop-Forward video. The scene contains a pedestrian walking from left to right, slowing and stopping for a few seconds at the center of the frame and walking out of the scene to the right. The resulting change point geometry is shown by the collection of change point vertices and change point hulls visualized in figures 4.20e and 4.20f where the color denotes hull membership.
4.21 Tracing the path of change point vertices created by a 1-d object that enters a scene traveling at a constant velocity from left to right, slows and stops, pausing for a time before accelerating and leaving the camera view at a constant velocity to the right. The path of change point vertices is shown in purple as a function of space and time. When the object enters the scene at relatively constant velocity, the vertices trace a line. As the object slows, the sequence of change point vertices curves upwards as less space is traveled per unit time. When the object stops, there are no additional change point vertices in the frame and the duration of the previous vertex is extended until the object begins to move again. Once the object begins to accelerate, new change point vertices trace a curve until a terminal constant velocity is reached and the vertices again follow a linear path.

4.22 Checker-Back video sequence. The scene contains specular metallic checkered background as seen in the example frames in figures 4.22a through 4.22d. The scene consists of a pedestrian which enters the camera’s view, initially traveling at a constant velocity from left to right. At approximately the frame center, the pedestrian slows to a halt and waits a few seconds prior to backing out of the scene in the direction he came from. Figures 4.22e through 4.22h visualize the change point vertices and change point hulls associated with the pedestrian in 3-d image space-time from different vantage points.
4.23 Idealized illustration of the change point geometry induced when imaging a 1-d that enters a scene with constant velocity, slows to a halt and pauses prior to accelerating to a constant velocity, exiting the scene from the direction it came. The sequence of change point vertices in 2-d space-time is shown in purple. As the object enters the scene with constant velocity, the change point vertices trace a line with positive slope. As the object slows, the change point vertices trace a curve as less space is traveled per unit time. While the object is at rest, no new change point vertices are created, leaving a gap in space-time. When the object accelerates to the left, the sequence of change point vertices trace a curve until a constant velocity is reached and the vertices trace a line. Due to the direction of the object, the latter linear set of vertices follows a line with negative slope compared to the line associated with the vertices created when the object entered the scene.

4.24 1-d example using the eigenvalues and eigenvectors as change point hull descriptors. The blue points correspond to change point vertices in 2-d space time that belong to the same change point hull. The red arrows are the eigenvectors of the change point vertex scatter matrix and their length is proportional to the magnitude of their corresponding eigenvalues. The eigenvector with the largest eigenvalue is parallel to the line that best fits the change point locations. The eigenvector with the smallest eigenvalue is perpendicular to the best fitting line.

4.25 Visualizing the direction of motion for one-dimensional objects. The green and blue dots correspond to change point vertices associated with the motion of two different 1-d objects. The green and blue arrows on the change points correspond to the eigenvector with smallest eigenvector computed using the scatter matrix of the locations of the change vertices in each group. The projection of the eigenvectors onto the camera plane is in the direction of object motion as visualized by the arrows just above the imaging line.
4.26 Visualizing the effects of object velocity on the magnitude of the projected eigenvectors. A 1-d object moving at a slow constant speed creates change point vertices that trace a vertical line compared to an object with greater lateral speed. The change point vertices associated with the first object, which is moving slowly, are visualized as green dots. The change point vertices associated with a second 1-d object moving quickly are visualized as blue dots. The projection of the normal of the best fitting lines with regards to both sets of change point vertices are shown right above the imaging line. The magnitude of the projected vectors is inversely proportional to object speed.

4.27 Motion vectors recovered from the Crossing-Paths video.

4.28 Walk-Forward region-level change detection results. The first row of images, figure 4.28a show example frames of video from the Walk-Forward sequence in order from left to right. Figures 4.28b, 4.28c and 4.28d visualize the change detection results of the proposed region-level, SGMM-SOD and STLBP algorithms respectively for the frame in the corresponding column. A white pixel represents a significant change and a black pixel does not.

4.29 Change detection results for region level algorithms using the Stop-Forward video. The first row of images, figure 4.29a show frames from the Stop-Forward video in order from left to right. Figures 4.29b, 4.29c and 4.29d visualize the change detection results of the proposed region-level, SGMM-SOD and STLBP algorithms respectively. Each change detection mask corresponds to the output for the frame in the same column. A white pixel indicates a significant change was detected in the corresponding location whereas a black pixel indicates no change was detected in that area.
Chapter 1

Introduction

Due to advances in digital device manufacturing, there exists a huge amount of high quality, networked cameras deployed for video surveillance. A high definition, night vision enabled, wirelessly networked digital camera system with dedicated storage can be purchased from online retailers for a few hundred US dollars. In addition to conventional fixed cameras, satellite imaging systems, Unmanned Areial Vehicles (UAVs) and common commercial grade "drones" equipped with high definition cameras collect massive amounts of video on a daily basis. In addition to law enforcement activities, this enormous amount of video data is constantly analyzed for environmental monitoring, urban planning, and military applications.

With cost ratios for processing and storing digital data approaching cents per petabyte, manual analysis is the most costly component of video surveillance. Even if manual video analysis was cost effective, it is not technically feasible given the large, constant stream of real-time video feeds. For example, while there is no definitive audit, an article published by The Guardian in 2013 estimates there are approximately 1.85 million Closed-Circuit Television (CCTV) cameras in the UK alone, approximately one camera for every thirty two people [24]. Figure 1.2a visualizes the distribution of CCTV cameras in London, estimating the number of cameras per one thousand people by local borough while fig. 1.2b reflects the distribution of cameras across UK, underscoring the prevalence
(a) A commercial high-definition surveillance system equipped with 4 Night Vision Cameras, 500 gigabytes of dedicated storage and real-time accessability via a 3G enabled smartphone. Available for approximately $188.00 USD at the time of writing.

(b) The infamous predator drone equipped with video surveillance technology for reconnaissance missions.

(c) A commercial drone with attached high definition video camera. Available for approximately $60.00 USD at the time of writing.

Figure 1.1: CCTV digital video surveillance system, military-grade Unmanned Aerial Vehicle (UAV) and a commercial drone equipped with high definition video cameras.

and vast coverage of modern video surveillance systems.

While governments and law enforcement agencies are engaged in constant video collection approaching full coverage of public spaces, it is possible for operators to monitor only a small fraction of the available data in real-time, relegating the role of video surveillance to deterrence and forensic evidence for after-the-fact analysis. Studies conducted in the UK on CCTV statistics estimate screen to camera ratios in the range of 1:4 to 1:30, a ratio of analysts to screens at approximately 1:16 where further studies estimate a human operative can monitor 1-4 screens at a time with 5-10 minutes of rest breaks away from the screen per hour [25].

The 2013 Boston Marathon bombings highlight the consequences arising from the disparity between the ability to collect and analyze surveillance video and motivate the effort to automate surveillance systems for law enforcement applications. On April 15, 2013, explosions occurred near the finish line of the Boston Marathon, killing three and injuring 264 people [26]. It took three days for the Federal Bureau of Investigation (FBI) to compile and perform an initial analysis of available surveillance video before they were able to release photographs and videos of suspects, turning to
(a) Number of CCTV cameras per 1,000 people (London boroughs)

(b) Number of CCTV cameras per 1,000 people by sample borough council

Figure 1.2: As one of the first western countries to adopt state-sponsored CCTV video surveillance systems, many studies concerning the prevalence and effectiveness of CCTV systems originate from the UK. Figure 1.2a shows estimates of the number of CCTV surveillance cameras per 1,000 people in London boroughs while figure 1.2b visualizes the number of cameras per 1,000 people across the UK. These statistics were compiled in 2009 with graphics taken from [10].

the public for help in uncovering the identities of the men wanted for questioning. Not only were the suspects successful in executing the bombing, but in the period of time where authorities were in the process of analyzing the available video evidence, the suspects killed an MIT policeman, car-jacked an SUV and were involved in a shootout with police.

A retrospective analysis of the available video data suggests the subjects behavior warranted closer inspection on the day of the bombing, had a manual operator been able to spot them. The suspects were both carrying large backpacks, dressed in similar attire while attempting to appear unacquainted yet were clearly following each other, maintaining a constant distance and seemingly uninterested in the marathon as a frame taken from surveillance video on the day of the bombing shows in fig. 1.3a. Additionally, video surveillance cameras were able to capture one of the suspects
planting the bomb, purposely leaving his backpack in the middle of a large crowd before fleeing the scene as seen in figure 1.3b.

Real-time analysis of video footage may have been able to stop the bombings or at least limit casualties and injuries as the act of planting the bomb was obvious upon review of the evidence. Fortunately, most people are not intent on committing acts of violence and most video captured by surveillance systems correspond to innocuous behavior. Unfortunately this results in a situation where operators are searching for the proverbial needle in a haystack; robust human analysis of huge streams of real-time video data is impossible, yet if it were, it might have saved many lives on April 15, 2013 and during the subsequent manhunt. The only solution is to augment manual analysis of video with autonomous systems to aid in real-time detection and prevention of violent crimes.

Figure 1.3: Video surveillance footage of Boston Marthon 2013 bombing suspects. Figure 1.3a shows suspects maneuvering the crowd to place bombs in the form of pressure cookers contained in their backpacks. Figure 1.3b is a frame of unreleased video footage compiled by the FBI showing one of the suspects, leaving a backpack in a crowded area before fleeing the scene. Images were taken from [11] and [12] respectively.

In addition to the technical obstacles rendering manual analysis of video surveillance systems untenable, there exists many documented cases of human operatives abusing surveillance systems including acts of voyerism, stalking, and illegal economic espionage [25, 27] rendering manual analysis of surveillance data undesirable. These abuses are not only limited to video surveillance data but are examples where human analysts were caught abusing expansive multi-modal systems such as
infiltrating private communications via emails listening to phone conversations in real-time. Such examples highlight the potential and severity of abuses regarding human analysis of surveillance data in general but also specifically concerning video surveillance data. Automated video security solutions not only have the potential to overcome the technical challenges associated with analyzing huge real-time data streams but limit the potential for civil rights abuses by replacing potentially exaustable, biased and curruptable human operatives with dispassionate, objective and indefatigable algorithmic analysis.

While current era is one of ubiquitous computing and digital information where storing, editing, and viewing digital images and video is trivial given the requisite resources, autonomously extracting useful information from video surveillance footage is an unsolved problem with profound socio-economic consequences. Currently, manual analysis of video streams reigns supreme as trust-worthy automation has not been achieved. The focus of this research is to advance the field of video surveillance by developing algorithms for detecting and localizing spatio-temporal regions of video data corresponding to abnormal events in monocular, fixed camera systems. This is known as the change detection problem within the Computer Vision community, a branch of science devoted to processing and understanding visual information.

Change Detection may be an end unto itself, e.g., as a monitoring tool in surveillance systems, or as a pre-processing step for further reasoning about video segments identified as meaningful. Commonly, normal scene dynamics are referred to as background while visual dynamics of interest are sometimes called foreground. While change detection algorithms abound, natural scenes are composed of complex, dynamic, stochastically periodic events re-occurring simultaneously over vastly different time scales, making it difficult for automated systems to distinguish between meaningful changes and normal scene dynamics. For example, variations of appearance due to forest growth change over decades, seasons induce visual fluctuations monthly, weather directly changes the appearance of a scene daily, and the transition from day to night causes hourly appearance changes. Figure 1.4 highlights the visual appearance shifts that take place even over the relatively short time
scale by stitching images taken of an urban scene at different times of a single day. Weather causes the cloud cover to change, the transition from day to night induces global intensity variations in addition to local artifacts such as street and building lights being turned off and on as day turns into night. While the appearance of the scene changes markedly over time due to multiple cyclic events, these changes are usually not of interest to security operatives, driving the need for algorithms which can learn a robust vocabulary for describing periodic events but are still sensitive to behaviors of interests to analysts.

Figure 1.4: Appearance of real-world scenes vary markedly over time with predictable epochs of recurring change. Image taken from [13].

To further compound the problem, it is impossible to define what a system should consider to be a relevant event without considering the needs of the end user; a malignant event in the context of one application may be benign in the context of another. For example, if the goal of a particular application is to count the number of cars entering and exiting a restricted area, it is necessary for the system to account for every car in the scene. However, if the system is to monitor a busy highway for irregular traffic activity such as a collision, then the system will need to consider common traffic patterns as normal and not declare appearance fluctuations associated with routine traffic activity as significant.

Many state-of-the-art change detection systems are designed to be unsupervised in that the system
designers attempt to program *apriori* a universal, scene invariant description of visual cues that correspond to coherent, significant events. This paradigm misplaces the responsibility of defining what is or isn’t a meaningful change on the system designer, not the end user. While it may be argued that unsupervised change detection systems provide a low-level foreground/background segmentation to be used as input for classification systems operating at higher levels of the semantic hierarchy, the underlying change detection system would still require customization to ensure the proper primitive information is passed to the next module. Unsupervised systems implicitly encode the designer’s definitions of meaningful change and are typically constrained by a target application or dataset. The goal of the research reported in this thesis is to design generic yet robust change detection algorithms that can be easily tailored to accommodate a large class of applications, each with potentially different definitions of meaningful and varying scene complexity.

While there exists a plethora of change detection algorithms, these methods admit a simple three class taxonomy based on the semantic level of the underlying features of the approach: pixel (low), region (mid), and object (high) level change detection algorithms.

Pixel or low level change detection algorithms [1, 28, 16, 29, 30, 5, 6, 31, 7, 8, 32] attempt to classify meaningful changes on a per-pixel basis by distinguishing rare from common intensity observations. While these algorithms tend to be computationally efficient and easy to parallelize, they lack knowledge of the spatial structure of appearance dynamics and as such fail to reliably describe and distinguish between meaningful and irrelevant scene dynamics. Pixel level methods tend to trade simplicity for a shallow vocabulary with which to describe coherent action or behaviors, requiring significant ad-hoc post-processing that is typically algorithm specific and fragile. Furthermore, as summarized in chapter 2, state-of-the-art low level algorithms categorically make unrealistic temporal assumptions, modeling pixel values instead of per-pixel intensity sequences. The per-pixel model and algorithms proposed in chapter 3 introduce a rich per-pixel temporal model for describing and recognizing periodic events.

Object (high) level change detection algorithms [14, 15, 33] operate on the opposite end of the
semantic spectrum, first attempting to recognize object categories present in a scene prior to reasoning about their behavior. While in theory this approach offers the most direct and rich vocabulary for describing scene dynamics, these systems suffer from severe practical limitations, requiring enormous amounts of training data per object category. For example, a generic object class such as a human may exhibit significant appearance variations under different lighting, camera resolutions or perspectives and normalizing for interclass variability restricts the ability to transfer object models to different applications and scenes. Further compounding the problem, it is impossible to define \textit{apriori} all object classes which may be observed in a video sequence, much less learn or build an accurate object model for each class.

For example, the work of [33] attempts to detect the presence of predefined activities in video such as "watching TV" or "making tea" using object recognition. This high level algorithm required labeling objects in a dataset of over 1 million frames of video for learning only 18 activities. The authors identify inter-class appearance variability as a source of significant difficulty, citing the example that a refrigerator looks very different when it is closed compared to when it is opened.

Standard unsolved object recognition challenges and benchmarks such as the PASCAL challenge [34] consist of almost 12,000 images for training and evaluating autonomous classification schemes for 20 object categories while the ambitious ImageNet [35] consists of over 14 million annotated images and 10,000 object classes. These datasets and challenges underscore the difficulties in modeling the appearance variability of even a small number of pre-defined object classes even in the current "big-data" era.

A measure of success has been achieved by high-level change detection algorithms that specialize on a single object class such as vehicles. In [14], Leotta and Mundy incorporate an explicit illumination model, 3-d vehicle model, and a hand-crafted 3 state Hidden Markov Model (HMM) for recognizing vehicles in surveillance video. The 3D vehicle model templates used in [14] are shown in figure 1.5a. The 3-d vehicle model of in [14] was augmented in [15] to include a deformable vehicle mesh based on the principal component decomposition of exemplar vehicles from the DARPA
(a) 3D vehicle models one for a truck, sedan and SUV.

(b) Deformable 3D vehicle model. A parameterized mesh allows a single model to deform into shapes corresponding to different types of cars.

(c) Training (E3D) data set.

Figure 1.5: Detecting vehicles in video surveillance. Figure 1.5a originally appeared in [14] and figures 1.5b and 1.5c are adopted from [15].

exploitation of 3-d data (E3D) data dataset, a large collection of 3-d CAD (Computer Aided Design) models, a sample of which is shown in figure 1.5c. However, this method has not yet been extended to accommodate multiple object classes and requires a large amount of high fidelity 3-d models for training the deformable vehicle model.

Because robust object detection and recognition is still an unsolved computer vision task, it is necessary for performant, state of the art change detection algorithms to operate within the low or
medium levels of the change detection taxonomy, at least for the foreseeable future.

Region (mid) level change detection algorithms [36, 9, 22, 37, 38, 39, 40, 41, 42] identify space-time features which are correlated to object behavior without explicitly localizing and classifying all objects in a scene. Commonly, statistics concerning configurations of region descriptors such as edges or gradients are learned during training and statistics collected in new video are compared to the model with low probability observations flagged as change [9, 22]. Other algorithms take a "puzzle" approach and attempt to reconstruct new video observations using a database of image patches collected from previously indexed video. Videos that cannot be reconstructed using the database entries with sufficient confidence are labeled as a change of interest [37, 38]. By incorporating region statistics, mid-level algorithms have the potential to learn a richer spatio-temporal vocabulary for describing scene dynamics and detecting changes. The region or mid level algorithm proposed in chapter 4 is a compromise between the efficient low-level and descriptive but impractical high-level approaches which uses the geometry of space-time discontinuities to localize and quantify coherent actions in video sequences.

1.1 Contributions and Outline

State-of-the-art change detection algorithms categorically suffer from models with strong temporal independence assumptions. Chapter 2 reviews modern pixel-level change detection algorithms focusing on the weaknesses of the temporal modeling and reasoning imposed by each. Chapter 3 proposes a per-pixel intensity sequence model based on a class of hidden semi-Markov models (HSMM), a duration dependent hidden Markov model (DDHMM) to accurately account for stochastically periodic phenomena prevalent in real-world video. State assignments inferred by a novel incremental learning algorithm are used as features for constructing per-pixel codebooks during a training phase to identify changes of interest in new video. Chapter 4 reviews prior region-level change detection algorithms and proposes an extension of the per-pixel algorithm derived in chapter 3, relaxing the
per-pixel spatial independence assumptions by exploiting the spatio-temporal geometry of DDHMM state transitions to identify coherent object behavior for detecting change.
Chapter 2

Pixel-level change detection

This chapter reviews state-of-the-art pixel-level change detection algorithms for video sequences taken from fixed cameras. Each section groups change detection algorithms based on the fundamental per-pixel observation model employed by each algorithm. Section 2.2 reviews Gaussian mixture model (GMM) based change detection algorithms including the Stauffer-Grimson algorithm (SG-GMM) which forms the foundation for many GMM based variants including the algorithm proposed by Heras et al. [9] and the Bayesian non-parametric GMM change detection algorithm proposed by Haines et al. [16, 29] in section 2.3. Section 2.4 discusses applications which apply Hidden markov models (HMMs), a first order temporal extension of the GMM model, to change detection. The methods proposed by Paragios [30] and Leotta [14] are reviewed followed by an examination of a key weakness of the HMM which motivates many of the developments proposed in this thesis. Non-parametric methods including Gaussian kernel density estimation (GKDE) based change detection algorithm [5] and sample distance methods ViBe [4, 3] and PBAS [2] are discussed in section 2.5. A visual codebook method which was influenced by the performance requirements of GKDE, the CBBGS proposed by Kim et al., is reviewed in section 2.6. Change detection algorithms that apply domain transformations, namely Wave-Back [7] and Waviz [8] are reviewed in section 2.7. For clarity, each algorithm has been given an abbreviation which is listed along with a summary of the
approach and primary sources in table 2.1.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Summary</th>
<th>Primary Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG-GMM</td>
<td>Stauffer-Grimson per-pixel Gaussian mixture model</td>
<td>[1]</td>
</tr>
<tr>
<td>DP-GMM</td>
<td>Dirichlet Process Gaussian mixture model</td>
<td>[16, 29]</td>
</tr>
<tr>
<td>GKDE</td>
<td>Gaussian Kernel Density Estimation</td>
<td>[5]</td>
</tr>
<tr>
<td>ViBe</td>
<td>Nonparametric, randomly updated sample history</td>
<td>[4, 3]</td>
</tr>
<tr>
<td>PBAS</td>
<td>Nonparametric, randomly updated sample history</td>
<td>[2]</td>
</tr>
<tr>
<td>CBBGS</td>
<td>Codebook based background subtraction algorithm</td>
<td>[6]</td>
</tr>
<tr>
<td>Wave-Back</td>
<td>Discrete Cosine transform of intensity sequences</td>
<td>[7]</td>
</tr>
<tr>
<td>Waviz</td>
<td>Fast Fourier Transform of intensity sequences</td>
<td>[8]</td>
</tr>
</tbody>
</table>

Table 2.1: Abbreviations and summary of reviewed change detection methods.

2.1 Preliminaries

2.1.1 Notation

The proposed algorithms and many of the reviewed prior works approach the change detection problem from a probabilistic or statistical point of view. Therefore, it is helpful to define the fundamental quantities, notations and conventions to unify the discussion algorithms that make use of probabilistic concepts. Fundamental to probability is the notion of a random variable: a variable that make take a value over a domain according to a probability distribution. Throughout this thesis, random variables will be denoted using an italicized capital letter while the corresponding lower case represents a specific instantiation of the corresponding random variable. Random variables may be continuous or discrete, uni- or multi-variate depending on the context. A probability distribution, also called a density, associated with a random variable, such as \( A \), is denoted \( p(A) \). Conditional probability distributions express the probability distribution of one random variable given the particular value of another. For example, consider the random variables \( A \) and \( B \), the conditional density of \( A \) given a particular value of the random variable \( B \) will be written \( p(A \mid B = b) \), \( p(A \mid B) \) or \( p(A \mid b) \) in the appropriate context.

Typically, a probabilistic model specifies the probability of a measurement which may depend on
the settings of a collection of controlling parameters $\Theta$ which are trained using historical samples or observations of the physical, real-world process that is to be modeled. A collection of ordered observations, commonly referred to as a dataset, will be denoted $\mathcal{Y} = \{y_1, y_2, \ldots, y_t\}$. The probability of a dataset with respect to a given probabilistic model and parameter configuration is commonly called the likelihood function of the model and is written as $L(y_1, y_2, \ldots, y_t | \Theta)$. It is typical to view the likelihood as a function of the model parameters not observations and is generally not a normalized probability distribution.

### 2.1.2 Generative Probabilistic Models

Generative probabilistic models (GPMs) refer to an important class of probabilistic models that admit procedures for generating synthetic observations. In general it is possible to fully specify a probabilistic model, including a GPM, by defining the model’s likelihood function. However, generative models admit simpler yet equivalent specifications. All generative models assume an underlying stochastic process is responsible for generating the observations associated with a real-world processes. While the underlying generative process does specify the recipe for generating synthetic datasets, it simultaneously fully defines the probabilistic model and underlying likelihood function. While it is common in standard literature for generative probabilistic models to be defined in terms of the likelihood function the likelihood function is typically presented as an esoteric equation. However, generative models typically admit simple procedures that clearly enumerate the key assumptions of the stochastic model which is assumed to govern the dynamics of the real-world phenomena. Therefore, when a generative model is introduced, the generative process which is assumed by the model will be specified followed by the likelihood function which may be derived from the generative specification by applying the basic rules of probability to the generative process. The goal is to first intuitively introduce the main properties and assumptions of the generative model and then subsequently present the traditional, more formal likelihood specification.
2.1.3 Probabilistic Graphical Models

Probabilistic models may be specified in graphical form granting two major advantages: complex distributions are expressed using intuitive diagrams rather than mathematical equations, and learning and inference algorithms may be more easily derived by inspecting a graphical structure rather than direct algorithmic manipulation [43, 44, 45]. An example of a graphical model is given by figure 2.1 which specifies the factorization of the joint probability distribution over two random variables, \( p(A, B) = p(A) p(B \mid A) \). The directed arrow represents a conditional dependence where the tail is conditioned on the source and random variables are represented as nodes in the graph with dotted outlines.

Figure 2.2a is an example which shows how the density of a collection of independent observations \( \{y_1, y_2, \ldots, y_T\} \), deterministic values, are incorporated into the graphical representation. Generally, clear nodes with dotted outlines will denote random variables and shaded nodes will represent observed data. Additionally, the lack of connections between the set of random variables and indexed observations specifies the \( T \) conditional distributions are independent. Therefore, the joint probability of the data and random variables is the product of each conditional density:

\[
p(y_1, y_2, \ldots, y_T, A_1, A_2, \ldots, A_T) = \prod_{t=1}^{T} p(y_t \mid A_t) \tag{2.1}
\]

Figure 2.2b is an equivalent, condensed representation of figure 2.2a using plate notation which specifies a number of independent replications of the template conditional relationship according to the integer or variable in the lower right corner of the plate.

Figure 2.1: Example of a probabilistic graphical model. This directed graphical model specifies the following conditional factorization of the joint probability distribution over random variables \( A \) and \( B \): \( p(A, B) = p(A) p(B \mid A) \).
(a) Example of a probabilistic graphical model (b) An equivalent representation of figure 2.2a corresponding to independent observations with using plate notation. The conditional dependent between an observation $y_t$ and the random variable $A_t$ is replicated $T$ times.

Figure 2.2: Specifying an independent factorization over a collection of observations which are conditionally dependent on a random variable. Random variables are shown as clear nodes with dotted outlines while observations are represented as shaded circles. Figure 2.2b is an equivalent, condensed representation of figure 2.2b using plate notation.

2.2 Gaussian mixture models

Figure 2.3: Visualizing a Gaussian mixture model using Probabilistic Graphical Models. Shaded nodes represent observed quantities and circles with dotted outlines are random variables.

One of the most widely adopted and studied change detection algorithms is the per-pixel Gaussian mixture model (GMM) proposed by Stauffer and Grimson [1] and is referred to as SG-GMM. Stauffer and Grimson model the intensity distribution at each pixel using independent Gaussian mixture models; a linear combination of $K$ weighted Gaussian distributions, where $K$ is an integer parameter
typically set between 3 and 5. In general, per-pixel observations may be vector valued, such as a vector of RGB color values or a scalar such as gray scale intensity. The parameter $\mu_k$ is the mean of the $k^{th}$ Gaussian component and $\Sigma_k$ is the covariance. If the dimensionality of the observations is $n$, $\mu_k$ is a vector of size $n$ and $\Sigma_k$ is a matrix of size $n \times n$. If $n = 1$, both parameters are scalars and the component dependent variance is denoted $\sigma_k$ rather than $\Sigma_k$.

Gaussian mixture models are a special case of a general class of models called *Finite Mixture Models* which assume a dataset $\{y_1, \ldots, y_T\}$ is generated according to the following random process:

**Generative Model 1. Finite Mixture Model**

1. For each time $t \in \{1, \ldots, T\}$
   
   (a) Choose a state $s_t \sim P(S)$
   
   (b) Choose an observation $y_t \sim p(Y \mid s_t)$

The general finite mixture specifies a set of components indexed by a hidden or latent random variable $S \in \{1, 2, \ldots, K\}$, otherwise known as a *latent state* which is not directly observed and the collection $\{s_1, s_2, \ldots s_T \mid s_t \in S\}$ catalogs a specific hidden state configuration. Given a particular state $s_t$, an observation at time $t$ is chosen according to a state conditional distribution called an *emission* or *appearance* distribution denoted $p(Y \mid s_t)$. A GMM is merely an instantiation of the general Finite Mixture Model where the emission distribution is assumed to be a *Gaussian* (also called *Normal*) distribution given by equation 2.3. The generative process for a GMM defines the likelihood function given by equation 2.2 where in general a likelihood function specifies the probability of observing a particular dataset and is typically viewed as a function of the model parameters $\Theta = \{\{\omega_k, \mu_k, \sigma_k\}, k \in \{1, \ldots K\}\}$.

$$L(y_1, y_2, \ldots, y_T \mid \Theta) = \prod_{t=1}^{T} \sum_{k=1}^{K} p(S = k)p(y_t \mid S = k, \mu_k, \Sigma_k) \tag{2.2}$$

$$p(Y = y \mid S = k) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(y_\mu_k)^T\Sigma_k^{-1}(y_\mu_k)} \tag{2.3}$$
\[ p(S = k) = \omega_k \text{ where } 0 \leq \omega_k \leq 1, \quad \sum_{k=1}^{K} \omega_k = 1 \] (2.4)

Equation 2.4 encapsulates the prior uncertainty (independent of observing any data) that a given state \( s_t \) will be chosen responsible for generating an observation. In the case of a GMM, the complete set of model parameters, \( \Theta = \{ \theta_1, \theta_2, \ldots, \theta_K \mid \theta_k = \{ \omega_k, \mu_k, \Sigma_k \} \} \), is a set \( K(n^2 + n + 1) \) numbers that fully specifying the model for an observation with dimensionality \( n \). Note the convention followed throughout this thesis: Capital letters denote random variables and lower case specific instantiations of the corresponding variable.

Stauffer and Grimson derive a incremental method for learning the per-pixel model parameters and an intuitive heuristic for detecting change. Given a sample of per-pixel observations \( \{y_1, y_2, \ldots, y_T\} \), learning refers to the process of optimally (according to some measure) choosing a configuration of the model parameters, \( \Theta \). The proposed incremental learning algorithm is based on Maximum Likelihood Estimation and incremental Expectation Maximization (EM) with exponential forgetting [46, 47]. Stauffer and Grimson argue that traditional EM update equations would be inappropriate for learning per-pixel GMMs parameters in the context change detection as the world is in a constant state of flux and changing appearance over time and instead propose a set of equations for learning that allow the model parameters to forget past observations and learn a new distribution based on more recent observations.

At a given pixel location, every new observation \( y_t \) is compared to each of the existing \( K \) Gaussian components. If the current observation is within 2.5 standard deviations of a given component, the observation is considered to "match" the component. If the observation does not match any of the components, the least probable distribution’s parameters are replaced by a mean centered on the observation with initially large variance (an algorithmic parameter) and low weight. In the case of an observation which matches multiple components, the matches are ranked by the distance between the observation and the mean of the component with the best matching component updated...
according to:

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1} + \alpha$$  \hspace{1cm} (2.5)$$

$$\mu_{k,t} = (1 - \rho)\mu_{k,t-1} + \rho y_t$$  \hspace{1cm} (2.6)$$

$$\Sigma_{k,t} = (1 - \rho)\Sigma_{t-1} + \rho(y_t - \mu_t)^T(y_t - \mu_t)$$  \hspace{1cm} (2.7)$$

$$\rho = \alpha p(y_t | \mu_k, \Sigma_k)$$  \hspace{1cm} (2.8)$$

These equations allow the parameters of each mixture distribution to change over time according to a specified learning rate $\alpha$, an additional global (applies to each pixel) parameter. Thus far the per-pixel GMM model has been developed for observations with arbitrary dimensionality $n$, SG-GMM considers gray scale intensity video where the variance of the resulting univariate distributions is denoted $\sigma_k$. To detect scene changes, Gaussian components are sorted in descending order according to the predicate $\frac{\omega_k}{\sigma_k}$ pushing components with a greater number of matching observations and smaller variances to the beginning of the sorted list. The first $B$ distributions with accumulated rate larger than a global threshold $\Omega$ are chosen as a model for normal scene appearances

$$B = \arg\min_b \left( \sum_{k=1}^b \omega_k > \Omega \right)$$  \hspace{1cm} (2.9)$$

If an observation matches with one of the first $B$ components, it is not considered a significant change while the converse is true if the observation does not match a component in $B$. A binary mask is created with a 1 representing a significant change and 0 representing normal scene dynamics for each frame. A post processing step consisting of the connected components algorithm described in [48] is used to fill gaps and remove noise in the per-pixel change detection mask.

An additional global parameter $\sigma_{min}$ is typically added to address a practical consequence of estimating Gaussian parameters using the EM algorithm and its variants. If a Gaussian components’ estimated variance is close to zero, $\sigma^2 \approx 0$, equation 2.3 to degenerates resulting in singularities. This is typically circumvented by placing a lower bound on the estimated value of $\sigma_{k,t}$ where $\sigma_{min}$ is substituted for the computed estimate if it falls below $\sigma_{min}$. 

While the algorithm proposed by Stauffer and Grimson may adopt to appearance changes over a short time scale, the underlying per-pixel intensity model accounts for typical pixel intensities, not intensity sequences; the order in which intensities are observed at a given pixel has no effect on the likelihood of the observation sequence for a fixed set of parameters. This property discards relevant temporal information and obscures the ability for reasoning about cause and effect of coherent behaviors in videos that have been encoded using the per-pixel GMM representation. A simple thought experiment consisting of a scene containing a traffic light shows the weakness of this model. The probability of a pixel being green, yellow or red is proportional to the total number of times each color is observed and the fact that there are always direct transitions from green to yellow, yellow to red and back to green cannot be captured by this model.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>Number of Gaussian Components</td>
</tr>
<tr>
<td>$\sigma_{init}$</td>
<td>New Gaussian component initial standard deviation</td>
</tr>
<tr>
<td>$\sigma_{min}$</td>
<td>Minimum Gaussian component standard deviation</td>
</tr>
<tr>
<td>$\sigma_{thresh}$</td>
<td>Observation-component matching threshold</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>Background component threshold</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Learning rate</td>
</tr>
</tbody>
</table>

Table 2.2: Summary of parameters for algorithms based on Gaussian Mixture Models as described by [1]

Heras et. al [9] build a coarse temporal background model using two GMMs, one with a slow and another with a fast learning rate, to determine if a pixel intensity belongs to a static or dynamic background. Stable edges in the scene are also tracked to account for objects that have entered or been removed from the frame, incorporating region-level information with the augmented per-pixel model. However, the short and long term learning rates must be specified \textit{apriori} and such a coarse quantization of time limits the systems ability to disambiguate events in the same scene that reoccur at different rates. Additionally, modeling independent intensity values at multiple temporal scales cannot capture information about temporal patterns or reason about cause and effect.
2.3 Model Selection and Bayesian Methods for Gaussian mixture models

Figure 2.4: Visualization of DP-GMM using probabilistic graphical model plate notation adapted from [16]. The $\alpha$ and $H$ random variables encapsulate the Dirichlet process prior while the $\mu$ and $\Lambda = \{n_k, \kappa_k, \sigma_k\}$ nodes represent the component dependent variables, taken to be random quantities within the Bayesian Framework. Latent component indicator variables $S_t$ are used to independently model pixel observations $y_t, t \in \{1, \ldots, T\}$. The process is replicated independently for each pixel in a frame of video $F$.

*Model complexity* typically refers to the cardinality of the parameter set $\Theta$ while the process of choosing an appropriate value of model complexity for a given problem is referred to as *model selection*. With respect to Gaussian mixture models, model complexity is controlled by the parameter $K$, the number of Gaussian components of the model. Significant research has been devoted to automatically inferring the appropriate number of Gaussian components $K$ from data rather than selecting the value manually *apriori*. The *Split Merge EM Algorithm* introduced in [49] is a general method for model selection for Gaussian Mixture Models which is applied to Change Detection by Heras et. al. [9] for incrementally computing a data driven estimate of $K$. Zivkovic [28] also proposes an extension of the original Stauffer-Grimson algorithm and introduces an incremental estimate of $K$ based on arguments about the estimated set of component weights. The model selection idea described in [28] is again implemented by the region-class change detection method proposed in [50].
A batch of recent Change Detection methods detailed in [51, 16, 29] borrow from the *Nonparametric Bayesian* literature [52, 53, 54] for estimating the complexity of pixel independent Gaussian Mixtures. In the context of the Bayesian framework, the term *nonparametric* does not imply the model is nonparametric but is instead a reference to underlying assumption that model complexity is not a deterministic parameter, but a random variable which may be estimated by observing samples of the underlying process. In general, The *Bayesian* paradigm assumes all model parameters are random variables in contrast to the traditional *Maximum Likelihood Framework* where model parameters are deterministic values to be estimated via optimization. Using the Bayesian formalism, model parameters are associated with a prior distribution, a probability density which models the belief of a particular parameter configuration prior to observing any data. Given new samples, prior density is refined and the updated density is called the *posterior* density. Bayesian learning is based on its similarity to Bayes’ Theorem which expresses how to alter the belief in an event \( A \) after observing the occurrence of another event \( B \):

\[
p(A | B) = \frac{p(B | A) p(A)}{p(B)}
\]  

\( (2.10) \)

Learning model parameters according to the Bayesian framework based on Bayes’ theorem where the posterior belief of the density governing a collection of model parameters \( \Theta \) is related to evidence expressed by the likelihood of a given data set \( D \) given a particular parameter configuration weighted by the prior belief in the configuration:

\[
p(\Theta | D) \propto L(D | \Theta) \cdot p(\Theta)
\]  

\( Posterior \), \( Likelihood \), \( Prior \)  

\( (2.11) \)

Significant computational savings is achieved if the posterior distribution of the parameters \( p(\Theta | D) \) maintains the same functional form as the prior distribution \( p(\Theta) \) when weighted by the likelihood of the observed data \( L(D | \Theta) \). Such prior-posterior pairs are known as *conjugate Priors* and the posterior parameters may be computed using a closed form formula of the observations and prior distributions.
Haines et al. [16, 29] place a Dirichlet process prior on a Gaussian mixture model to automatically estimate the number of Gaussian components required to accurately model the color distribution at pixel location in a video sequence. The per-pixel Dirichlet process Gaussian mixture model (DP-GMM) is most easily understood by exploring the DP-GMM generative process assumed independently for each pixel location:

**Generative Model 2. DP-GMM**

1. For each datum \( y_t, t = \{1, 2, \ldots, T\} \)

   (a) with probability \( \frac{\alpha}{\alpha + t - 1} \) sample observation \( y_t \sim N(\mu_0, \sigma_0) \)

   (b) with probability \( \frac{n_k}{\alpha + t - 1} \) sample an observation \( y_t \sim N(\mu_k, \sigma_k) \) from an already existing component

where \( N(\cdot) \) is the Gaussian distribution of equation 2.3, \( n_k \) is the number of observations associated with the \( k^{th} \) Gaussian component and \( \alpha \) is a positive scalar parameter which controls the prior preference of the Dirichlet process to generate a new Gaussian component. Haines et. al. model each color channel independently using a conjugate Gaussian-Gamma distribution where the posterior mean and standard deviations have the form

\[
p(\sigma_i^{-2}) = \Gamma\left(\frac{n_{i,0}}{2}, \frac{\sigma_i^2}{2}\right) \quad p(\mu_i | \sigma_i^2) = N\left(\mu_{i,0}, \frac{\sigma_i^2}{\kappa_{i,0}}\right)
\]

where \( i \in \{0, 1, 2\} \) indexes the color channel and \( \mu_{i,0}, \sigma_{i,0}^2, n_{i,0} \) and \( \kappa_{i,0} \) are the parameters of the prior distribution, commonly called hyperparameters in the Bayesian framework. The graphical model expressing the Dirichlet process Gaussian mixture model is shown in figure 2.4 adapted from [16]. The \( \alpha \) and \( H \) nodes denote the Dirichlet process prior and \( \mu \) and \( \lambda \) encapsulate the component dependent parameters.

The priors for new Gaussian components are computed using global color statistics of the current frame,

\[
\mu_{i,0} = \frac{1}{|F|} \sum_{y_i \in F} y_i \quad \sigma_{i,0} = \frac{1}{|F|} \sum_{y_i \in F} (y_i - \mu_{i,0})^2
\]
where \( F \) is the set of pixels in a video frame, \(|F|\) is the number of pixels in the frame, and \( n_{i,0} = \kappa_{i,0} = 1 \) is constant. An incremental learning algorithm based on a Gibbs sampler is proposed by [16, 29] which argues that the large stream of per-pixel data available in video will result in rapid parameter convergence to nearly optimal settings. Essentially, a set of existing \( K \) Gaussian components is maintained in addition to a new hypothetical component \( K + 1 \) with a prior mean and standard deviation. If an observation is associated with an existing Gaussian component, the posterior parameters of the component are updated. If the new component is selected, its parameters are also updated, and the component is appended to the set of existing components. Component membership is sampled according to the predictive probability of the current observation, \( y_t \), given the background model, \( bg \).

\[
p(y_t \mid bg) = \frac{\pi_k}{\sum_{j=1}^{K+1} \pi_j} \prod_{i \in \{0,1,2\}} T \left( y \mid n_{t,i}, \mu_{t,i}, \frac{\kappa_{t,i} + 1}{\kappa_{t,i} n_{t,i}} \sigma^2_{t,i} \right)
\]

(2.14)

where \( T(\cdot) \) is a student-t density function [16, 29]. \( \pi_k \) represents the probability of component \( k \) computed as described in the DP-GMM generative model specification and the \( K + 1 \) component is the dummy new component. The parameters of the sampled component’s posterior is updated

\[
n_{i,m+1} = n_{i,m} + \omega, \quad k_{i,m+1} = k_{i,m} + \omega, \\
\mu_{i,m+1} = \frac{\kappa_{i,m} \mu_{i,m} + \omega y_i}{\kappa_{i,m} + \omega}, \quad \sigma^2_{i,m+1} = \sigma^2_{i,m+1} + \frac{\kappa_{i,m} \omega}{\kappa_{i,m} + \omega} (y_i - \mu_{i,m})^2
\]

(2.15)

where \( \omega \) is a sample dependent weight and \( m \) indexes the number of samples previously assigned to the component.

In contrast with the Stauffer-Grimson classification scheme which associated select Gaussian components with a background model and the remainder as foreground, Haines et. al. propose to solve a probabilistic binary classification on a regular lattice with spatial regularization detect change. The goal is to generate and optimal labeling for each pixel location in a new frame of video as either foreground \( fg \) or background \( bg \) according to \( p(bg \mid y) \), the probability of the background
class conditioned on an observed pixel value. By Bayes’ Theorem:

$$p(bg \mid y) = \frac{p(y \mid bg)p(bg)}{p(y \mid bg) + p(y \mid fg)}$$  \hspace{1cm} (2.16)$$

where the per-pixel DP-GMM models $p(y \mid bg)$ while $p(bg)$ and $p(x \mid fg)$ are unknown. $p(bg)$ is taken to be an implicit threshold set to 0.5 and it is argued that $p(x \mid fg) = 1$ is an appropriate value assuming a uniform distribution over a normalized color space. Spatial regularization is achieved by defining an inter-pixel label similarity function that respects the hypothesis that neighboring pixels typically maintain the same label (foreground or background) as their neighbors. The per-pixel probability and pixel label similarity weight are used to construct a binary labeling problem on a regular lattice which is solved using belief propagation [55].

While the resulting algorithm is an elegant application of Bayesian Nonparametrics, it is difficult to identify the components of the algorithm proposed in [16, 29] responsible for the reported performance gains relative to the original Stauffer-Grimson algorithm. Is the DP-GMM truly driving a more accurate per-pixel density estimation leading to enhanced results? The algorithm also implements a normalized color space coupled with an explicit lighting change model along with spatial regularization; are these additions responsible for the performance gains observed with respect to [1]? These questions can only be answered with further studies, yet it is clear that the per-pixel DP-GMM model still suffers from the same fundamental limitations as the original Stauffer-Grimson algorithm. The underlying per-pixel model is temporally agnostic and phenomena which give rise to appearances that oscillate with large delay but persist for a short amount of time will ultimately be associated with a Gaussian components with small component probability, ultimately receiving a low per-pixel probabilistic appearance score. The only way to model such phenomena is to explicitly account for likely transitions between components conditioned on typical component durations.
2.4 Hidden Markov Models

Hidden Markov Models (HMMs) are a first order relaxation of temporal independence assumptions imposed by a Gaussian Mixture Model. Like Gaussian Mixture Models, HMMs utilize a finite number \((K)\) of component dependent probabilistic models, again called emission densities. However, HMMs incorporate knowledge about likely transitions between components and assumes the following generative process:

**Generative Model 3. HMM**

1. Choose a component \(s_1 \sim p(S)\)
2. Choose an observation \(y_1 \sim p(y \mid S = s_1)\)
3. For \(t \in \{2, \ldots, T\}^{a}\):
   
   (a) Choose a component \(s_t \sim p(S_t \mid s_{t-1})\)
   
   (b) Choose an observation \(y_t \sim p(y \mid S = s_t)\)

Equivalently, an HMM is defined by the following joint probability of observations and latent state configurations:

\[
p(y_1, \ldots, y_T, s_1, \ldots, s_T \mid \theta) = p(y_1 \mid s_1)p(s_1) \prod_{t=2}^{T} p(y_t \mid s_t)p(s_t \mid s_{t-1})
\]

where \(p(s_1)\) is an initial state distribution, \(p(s_t \mid s_{t-1})\) is a state transition distribution and \(p(y_t \mid s_t)\) is the familiar state conditional emission distribution. The likelihood function for a HMM would be computed by marginalizing equation 2.17 with respect to all possible latent state configurations.

\[
\mathcal{L}(y_1, \ldots, y_T \mid \Theta) = \sum_{s_1, \ldots, s_T} p(y_1, \ldots, y_T, s_1, \ldots, s_T)
\]

Due to the introduction of the transition distribution, \(p(s_t \mid s_{t-1})\), the summation in equation 2.18 does not factor independently for each observation like the GMM likelihood and brute force evaluation would require \(O(K^T)\) operations, growing exponentially in the number of observations.
However, it is possible to evaluate equation 2.18 in linear time using dynamic programming in linear $O(T)$ time [56, 45].

A common choice for the emission density is again the Gaussian distribution given by equation 2.3. The transition distribution is commonly a matrix $A \in \mathbb{R}^{K \times K}$, a table of $K \times K$ numbers with elements $A(i,j)$ appropriately normalized to represent the probability of transitioning from state $i$ to $j$. Assuming a HMM with $K$ states, Gaussian emission densities with dimensionality $n$ and a $K \times K$ transition matrix, $\theta$ is the set of all parameters of the HMM, a collection of $K(n^2 + n + K + 1)$ numbers $\theta = \{\pi, A, \theta_1, \ldots, \theta_K \mid \theta_k = \{\mu_k, \Sigma_k\}, \pi = \{\pi_1, \ldots, \pi_K\}, A \in \mathbb{R}^{K \times K}\}$.

Hidden Markov Models are an extension of the Finite Mixture Model explored in section 2.2 for sequential data; the order of the observed data modifies the value of the likelihood function defined by equation 2.18 in stark contrast with the GMM likelihood function given by equation 2.2. This point is not to be confused with the time varying nature of the parameter estimates proposed by Stauffer and Grimson [1]. Stauffer and Grimson’s learning algorithm vary the probabilities of successive observations not as a consequence of the per-pixel model but as an artifact of the heuristics imposed by the authors to account for the GMMs shortcomings in modeling a world undergoing constant appearance changes.

Hidden Markov Models are a logical extension of GMMs and have been previously used for modeling per-pixel intensity sequences in change detection applications. One of the earliest papers to employ a per-pixel HMM to model gray scale pixel intensity sequences was presented by Rittscher et. al. [57]. Rittscher et. al. first attempt to construct an HMM with three states for explicitly computing the per-pixel probability of foreground, background or shadow for traffic video. Foreground detections were then passed to a particle filter designed to track cars.

Leotta and Mundy [14] also utilize a three state Hidden Markov Model applied to the RGB channels of video; one state for background appearance, one for shadows and one for foreground that, due to the traffic monitoring application, was assumed to correspond to a vehicle. Rather than the typical Gaussian emission density for each state, a GMM emission density was used, admitting
multimodal state conditional densities.

Pargios et. al. [30] employ a per-pixel HMM for the general change detection problem and develop a data-driven objective function to incrementally infer HMM model complexity (the appropriate number of states of the HMM). However, a HMM only considers local temporal information, the next state probability distribution at any given time depends only on the previous state, and implicitly restricts state duration distributions to the geometric family. As experiments in sections 3.4 and 3.6 show, the temporal independence assumptions imposed by the HMM inhibit its ability to account for stochastically periodic phenomena typically found in video surveillance footage [56, 17].

For example, due to the temporal independence assumptions of the Hidden Markov Model, the probability of a latent state persisting for time $t$ is computed according to:

$$p(s_{1:t} = k) = p(s_{t+1} \neq k \mid s_t = k) \prod_{j=1}^{t} p(s_j = k \mid s_{j-1} = k)$$

$$= (1 - A(k, k)) \prod_{j=1}^{T} A(k, k)$$

$$= A(k, k)^t(1 - A(k, k))$$ \hspace{1cm} (2.19)

where $\prod_{j=1}^{t} A(k, k)$ is the probability of choosing state $k$ for $t$ independent draws before choosing any other state besides $k$ with probability $1 - A(k, k)$. Equation 2.19 is the definition of a geometric density with rate parameter $A(k, k)$.

Figure 2.5 plots the probability density of a geometric random variable for various rate parameters. The geometric density requires significant weight be placed on early transitions as larger values of $A(k, k)$ approach a uniform distribution for no other reason than a long state persistence. For example, consider a phenomena which persists for 80 time steps with no variance. An implicit geometric distribution results in a large rate parameter, necessarily inducing an artificially flat density even though the phenomena admits no uncertainty. It could be argued that the temporal scale of the density could be adjusted such that a single step of the density correspond to 80 time steps. Rather than introduce a separate module for estimating the natural time scale of state duration, a
simpler, more direct approach is to explicitly model state duration.

Augmenting a HMM with an explicit state duration density results in a class of models known as *Hidden Semi-Markov Models* (HSMM) or *Duration Dependent Hidden Markov Models*. While DDHMMs have been previously used in computer vision applications [58, 59, 60], they are typically used for high level, activity recognition or event detection to assign a global activity label to a frame of video or pre-segmented spatio-temporal region of interest. The model and algorithms proposed in chapter 3 use DDHMMs as a bottom-up, pixel level model of intensity sequences.

### 2.5 Nonparametric Pixel Intensity Models

Low-level *nonparametric* change detection algorithms refer to a class of methods that model intensity observations using a collection of previously observed values rather than a set of parameterized functions. In this context, *nonparametric* does not imply that the algorithm is devoid of parameters. In fact the contrary is typically true, the methods reviewed here show that there are a large number of parameters which control the change detection algorithm and the pixel level models.
2.5.1 Kernel Density Estimation

An early nonparametric method proposed by Elgammal et. al. [5] employed a Gaussian kernel density estimate (GKDE) to model intensity distributions for detecting low probability observations at each pixel location in a frame of video. In general, given a history of samples from a target distribution, \( \{y_1, \ldots, y_M\} \), a generic Kernel Estimator is a function of the form:

\[
p(y_t \mid \{y_1, \ldots, y_M\}) = \frac{1}{M} \sum_{m=1}^{M} \kappa(y_t - y_m)
\]

(2.20)

where \( \kappa \) is a scalar function of the observations. Elgammal et. al. propose a Gaussian or Normal kernel density estimator \( \mathcal{N}(0, \sigma) \) where \( \sigma \) is known as the bandwidth of the kernel:

\[
p(y_t \mid \{y_1, \ldots, y_M\}) = \frac{1}{M} \sum_{m=1}^{M} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(y_t - y_m)^2}{2\sigma^2}}
\]

(2.21)

Equation 2.21 may be interpreted as placing a Gaussian distribution (recall equation 2.3) centered on each historic sample with variance equal to the global bandwidth parameter. The probability of a new observation \( y_t \) is then the average of the densities associated with each sample. Figure 2.6 visualizes the density estimation process using Gaussian kernels. In these experiments, the true distribution was assumed to be a finite mixture of Gaussians with two components:

\[ \theta = \{\omega_0 = 0.4, \mu_0 = 25, \sigma_0 = 20, \omega_1 = 0.6, \mu_1 = 200, \sigma_1 = 10\} \]

which is drawn in each figure as the solid green line. Gaussian kernels centered on each sample drawn from the true distribution are shown as the dotted black lines in each plot and for visualization purposes, an artificial, fixed bandwidth and is used to maintain a uniform kernel height across plots. The density estimate, computed according to equation 2.21, is shown as the dotted magenta line. The kernel bandwidth \( \sigma \) is a model parameter and each row plots the estimate of the same sample set along each column using a different value of \( \sigma \). As \( \sigma \) increases from top to bottom, it is clear that larger bandwidths increase the smoothness of the estimate. To investigate how the estimate becomes more accurate for a given number of samples, the sample size \( M \) increases for each column from left to right for a fixed bandwidth (row). Elgammal et. al. argue that an appropriate per-pixel bandwidth may be estimated by
data and propose a bandwidth which is a function of the median of the absolute difference between samples:

\[
\sigma = \frac{\lambda}{0.68\sqrt{2}}.
\]

(2.22)

where \(\lambda\) is the median of \(|y_t - y_{t-1}|\).

While the per-pixel model proposed by [5] offers a non-parametric alternative to the per-pixel GMM outlined in section 2.2, the nonparametric Gaussian kernel density estimate still suffers from
the same weakness as a GMM; GKDE is unable to model temporal patterns. For example, figure 2.7 shows frames of surveillance footage taken of a restricted parking area with wind turbines in the distance. The corresponding intensity sequence corresponding to pixel highlighted with the red circle is plotted as a function of time (frames) in figure 2.8a. The majority of observations are associated with the intensity of the sky with a gray level of approximately 155 while the white blade of the turbine is observed at regular intervals with a corresponding gray level centered around 225. The intensity samples drawn in blue were used during a "Training" phase, consisting of the first 500 frames of video, to build the pixel intensity Gaussian kernel density estimate proposed by Elgammal et. al. [5] using the kernel bandwidth estimator given by equation 2.22.

The green line in figure 2.8a are observations taken from the video sequence used for "Testing" (frames 500-1000) the pixel intensity model, visualized in figure 2.8b. Gaussian kernels centered at training intensity observations, plotted as blue dots, are shown as black dotted lines. The aggregated density estimate, computed according to equation 2.21 is shown as the dotted magenta line. The majority of training observations are gray scale values associated with the sky appearance, as such there is a strong peak in the estimated density as the blade intensities occur with much less relative frequency. Figure 2.8 again shows the test intensity sequence as the green line which takes values with respect to the left axis. The red line in the same figure is the probability of each sample of the green line according to the learned kernel density model. The probability of intensity values similar to the sky are plotted with respect to the right axis and are very high. Notice observations corresponding to the color of the wind turbine blade are essentially zero and the blade is considered an extremely
(a) Intensity Sequence of Fields Point surveillance footage of pixel highlighted in red as shown in figure 2.7. The blue segment was used to train the Gaussian KDE estimate while the green segment was used to test the probability of observations given the model in figure 2.8c.

(b) Visualizing the Gaussian kernel density estimate of pixel intensity as proposed by Elgammal et. al. [5]. Black dotted lines are gaussian kernels centered at samples taken from the training period, shown as red dots. The estimated density is plotted as the magenta dotted line.

(c) Computing the probability of future observations given the learned per-pixel model. The observation sequence is plotted in green with respect to the left axis. The likelihood under the kernel density estimate is drawn in red with respect to the right axis.

Figure 2.8: Per-pixel Gaussian Kernel Density estimate proposed by Elgammal et. al. [5] for the pixel location highlighted in figure 2.7.

rare event under the GKDE model. Qualitatively, looking at the time series in figure 2.8a, this is simply not the case. The sky persists for a typical duration and the turbine blade reoccurs with
regularity. The per-pixel model proposed by Elgammal et. al. is unable to account for temporal information and therefore fails to recognize the simple sky-blade-sky intensity patterns.

### 2.5.2 Sample Distance Methods

The approach taken by the nonparametric algorithms ViBe [4, 3] and PBAS [2] is to explicitly store a history of $N$ exemplar appearance values at every pixel location. New observations are compared to the set of exemplars and both algorithms define heuristics for updating the fixed exemplar memory and similarity measures between observation samples with classification thresholds for segmenting foreground from background. The performance of ViBe and PBAS is greatly influenced by the history size parameter $N$, it must be set prior to deployment and implicitly sets an upper bound on the periodic dynamics the algorithm can associate with the background. Moreover, a background model represented by an unordered pool of pixel-level samples cannot be queried by higher-level algorithms to provide information about the temporal semantics of the background.

Further examination of the PBAS algorithm is helpful to clarify the fundamentals of nonparametric methods while investigating the mechanisms which cause these algorithms to fail where the proposed method succeeds. PBAS uses two per-pixel thresholds, $R$ a per-pixel decision threshold and $T$, an adaptive parameter controlling the rate at which the background model is updated. Both $R$ and $T$ are incrementally adapted to the scene dynamics observed at each pixel, requiring processing and storing $2|F|$ numbers where $|F|$ is the number of pixels in a single frame. The following decision rule classifies a single intensity value observed a given pixel, $I$, as change or not change:

$$F = \begin{cases} 
1 \text{ if } \# \{\text{dist}(I, B_k) < R\} < \#_{\text{min}} \\
0 \text{ else }
\end{cases}$$

(2.23)

$\# \{\cdot\}$ denotes the number of elements satisfying the enclosed condition, $\text{dist}(\cdot, \cdot)$ is defined by PBAS as a function of an observed pixel color and gradient magnitude, $\{B_1, \ldots, B_N\}$ is the current list of historic exemplars, $F = 1$ denotes a significant change with $F = 0$ the converse, and $\#_{\text{min}}$ is a global
parameter (applies to all per-pixel classifiers) set prior to run-time. PBAS asserts that the exemplar history set should only be updated with observations that correspond to normal scene dynamics, not significant changes to avoid false negative detections. That is, for an intensity observation to be considered as a potential exemplar it first must be classified as $F = 0$ according to the decision rule in 2.23. This is a key assumption, deserving special note for analysis proceeding the remaining details of the algorithm:

**Assumption 1.** For an intensity observation to be considered for inclusion in the exemplar list, it must be classified as $F = 0$ according to decision rule 2.23

When processing a new frame of video, for every pixel location, a random index $k \in \{1, \ldots, N\}$ is chosen and the exemplar $B_k$ is replaced with the current observation with probability $\frac{1}{T}$. In addition to storing a list of exemplars at each pixel, a list of $N$ minimal distances is maintained: $D = \{D_1, \ldots, D_N\}$. When an observation is added to the exemplar list, the minimal distance between the new exemplar and the previously existing exemplars is added to the minimum distance list and the mean minimal distance $\bar{d}_{min}$ is computed according to

$$\bar{d}_{min} = \frac{1}{N} \sum_k D_k.$$  \hspace{1cm} (2.24)

The mean minimal distance will be large if the exemplars exhibit significant appearance variations and will approach zero if the exemplars maintain a nearly constant value. The decision threshold $R$ is updated according to:

$$R = \begin{cases} R \cdot (1 - R_{inc/dec}) & \text{if } R > \bar{d}_{min} \cdot R_{scale} \\ R \cdot (1 + R_{inc/dec}) & \text{else} \end{cases} \hspace{1cm} (2.25)$$

where $R_{inc/dec}$ and $R_{scale}$ are additional algorithmic parameters.

The per-pixel learning parameter $T$ is designed to minimize mistakes made by misclassifying an observation as change, $F = 1$, that should have been associated with the background, $F = 0$ and considered as a potential exemplar. $T$ is therefore updated according to
Figure 2.9: Visualizing the estimated values of the decision threshold $R$ as computed by PBAS where larger values of $R$ are visualized are more white and smaller black in the left frame. The leaves blowing in the wind shown in the right figure yield higher variance in the minimal distance, inducing large values of $R$ compared to the pixels associated with the street or static objects in the scene. Image taken from [2].

$$T = \begin{cases} T + \frac{T_{inc}}{\bar{d}_{min}} & \text{if } F = 1 \\ T - \frac{T_{dec}}{\bar{d}_{min}} & \text{if } F = 0 \end{cases}$$ \hspace{1cm} (2.26)

where the parameters $T_{inc}$ and $T_{dec}$ are accompanied by two more global parameters, $T_{lower}$ and $T_{upper}$, which bound the range of $T$, $T_{lower} \leq T \leq T_{upper}$. The amount that the current value of $T$ is adjusted is inversely proportional to the appearance variability as captured by the minimum distance statistic $\bar{d}_{min}$; for large values or highly dynamic scene content, $T$ remains stable and changes more quickly for static content.

While PBAS is successful at detecting changes in many cases, Assumption 1 combined with heuristic 2.26 lead to undesirable effects concerning scenes containing periodic phenomena. For example, again consider the video surveillance footage monitoring a parking lot shown in figure 3.22. Focusing on a single pixel near the turbine, a typical high-level action sequence would be \{sky, blade, sky, \ldots\} where the sky appearance lasts for a long time relative to the short duration of the blade. The appearance history is therefore inundated with sky appearance exemplars with little or no exemplars representing the blade appearance. The relative homogeneity of the sky state will inevitably lead to a small value of $\bar{d}_{min}$, where the next observation of the turbine blade would
most likely be classified as meaningful change ($F = 1$) according to decision rule 2.23. Consequently varying $T$ according to equation 2.26, the dynamic value of $T$ would increase dramatically after classifying a blade observation’s appearance as foreground leading to a small probability $p = \frac{1}{T}$ that any subsequent blade observations would replace a sky exemplar even if a blade observation was classified as background ($F = 0$).

Droogenbroeck [3] improves upon the original [4] ViBe algorithm stating, ”one of the major difficulties related to the use of sample-based models is the handling of multimodal background distributions” and introduces a heuristic for detecting ”blinking pixels”, pixels locations which observe short time scale periodic phenomena. A ”blinking score” restricted to the range [1, 150] is maintained where the score is increase by fifteen or decreased by one according to a series of conditions. While the blinking heuristic helps suppress false positives associated with regular observations resulting from fast, regularly occurring appearances, ViBe still cannot learn long term temporal patterns and fails to associate the regularity of the wind turbine with the background. ViBe is similar to PBAS in that it maintains a fixed sized memory of exemplar pixels to compare against for detecting scene changes. However, ViBe defines it’s own set of heuristics which attempt to encode a universal definition of change and the reader is referred to [4] and [3] for the remaining details.

Tables 2.3 and 2.4 enumerate the parameters with default settings for PBAS and ViBe as determined experimentally and reported by [2] and [3] respectively. Despite the nonparametric classification, the large amount of free algorithmic parameters is typical of nonparametric models which rely on interconnected heuristic subprocesses. The parameters suggested and reported in tables 2.3 and 2.4 were used in all experimental comparisons.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Summary</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Number of historical exemplars</td>
<td>35</td>
</tr>
<tr>
<td>$#_{\text{min}}$</td>
<td>Number of components closer than $R$ be considered background</td>
<td>0.2</td>
</tr>
<tr>
<td>$R_{\text{lower}}$</td>
<td>Lower bound decision threshold</td>
<td>18</td>
</tr>
<tr>
<td>$R_{\text{scale}}$</td>
<td>Decision rule scale factor</td>
<td>5</td>
</tr>
<tr>
<td>$T_{\text{dec}}$</td>
<td>Rate to decrease $T$ if $F = 0$</td>
<td>0.05</td>
</tr>
<tr>
<td>$T_{\text{inc}}$</td>
<td>Rate to increase $T$ if $F = 1$</td>
<td>1.0</td>
</tr>
<tr>
<td>$T_{\text{lower}}$</td>
<td>$T$ lower bound</td>
<td>2</td>
</tr>
<tr>
<td>$T_{\text{upper}}$</td>
<td>$T$ upper bound</td>
<td>200</td>
</tr>
</tbody>
</table>

Table 2.3: List of parameters for PBAS with summary and parameter values suggested by [2] and used in all experiments.

<table>
<thead>
<tr>
<th>ViBe Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial update factor</td>
<td>1</td>
</tr>
<tr>
<td>Update factor (after initialization)</td>
<td>5</td>
</tr>
<tr>
<td>Number of samples per pixel (analogous to PBAS parameter $N$)</td>
<td>20</td>
</tr>
<tr>
<td>Number of required matches (analogous to PBAS parameter $#_{\text{min}}$)</td>
<td>2</td>
</tr>
<tr>
<td>Amplitude multiplicative factor</td>
<td>0.5</td>
</tr>
<tr>
<td>Amplitude matching threshold range</td>
<td>[20,40]</td>
</tr>
<tr>
<td>Color distortion threshold</td>
<td>20</td>
</tr>
<tr>
<td>Edge inhibition</td>
<td>50</td>
</tr>
<tr>
<td>Blinking value range</td>
<td>[0,150]</td>
</tr>
<tr>
<td>Blinking increment</td>
<td>50</td>
</tr>
<tr>
<td>Blinking decrement</td>
<td>15</td>
</tr>
<tr>
<td>Blinking threshold</td>
<td>30</td>
</tr>
<tr>
<td>Minimum size of foreground</td>
<td>50 pixel</td>
</tr>
<tr>
<td>Minimum size of foreground blobs</td>
<td>10 pixels</td>
</tr>
<tr>
<td>Minimum size of holes in foreground</td>
<td>20 pixels</td>
</tr>
</tbody>
</table>

Table 2.4: Enumerate parameters of ViBe with empirically determined default values suggested by [3] which were used in all experiments. The first four parameters were documented in the original ViBe paper [4], the remaining were added as improvements in [3].

## 2.6 Codebook based Pixel Intensity Models

While still considered a nonparametric change detection algorithm, CBBGS uses a codebook, a collection of visual codewords, rather than storing raw exemplar observations as a background model [6].

In fact, the CBBGS algorithm is similar to the general $K$-means clustering framework [61] where samples are represented as members of exemplar clusters or groups which partition a particular domain. CBBGS partitions the RGB color space of per-pixel observations by creating a codebook consisting of exemplar codewords, which are learned during a "training" phase. The codebook
approach was specifically developed by Kim et al. to address the memory requirements of other sample based nonparametric algorithms such as the GKDE algorithm outlined in section 2.5.1 that maintain a potentially large list of raw observations. A single codeword is able to simultaneously encode multiple observations, achieving a compressed representation compared with exemplar based methods. The authors compare the performance of CBBGS method to the Stauffer and Grimson (SG-GMM) [1] outlined in section 2.2 and the Gaussian Kernel Density Estimation (GKDE) algorithm [5] from section 2.5.1. Each algorithm was run on a common video sequence consisting of 1000 frames. The number of Gaussian components for the Stauffer Grimson algorithm was set to 10 while the per-pixel memory of the GKDE method was set to 300 samples. Table 2.5 shows that at these parameter settings, CBBGS is the faster than the SG-GMM algorithm for training and change detection, on par with GKDE for training and faster for change detection while achieving the smallest memory footprint in the experiment.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Storage (bytes)</th>
<th>Training (frames per second)</th>
<th>Classification (frames per second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBBGS</td>
<td>112</td>
<td>39.2</td>
<td>30.7</td>
</tr>
<tr>
<td>SG</td>
<td>200</td>
<td>8.3</td>
<td>12.1</td>
</tr>
<tr>
<td>GKDE</td>
<td>900</td>
<td>40.8</td>
<td>11.1</td>
</tr>
</tbody>
</table>

Table 2.5: Performance comparison of processing and memory requirements for CBBGS, SG [1] and GKDE [5] as reported in [6]. Note that for a given parameter setting, the memory requirements of the SG and GKDE algorithms are fixed. However, due to the adaptive codebook construction, the CBBGS memory requirements will change depending on observed scene dynamics. The value reported for the storage requirement of CBBGS is the computed with respect to the average codebook size which for the test scene was reported to be four codewords.

A unique codebook, $C$, is associated with each pixel location in a frame of video. Codewords, $\{c_l \in C \mid l = \{1, \ldots, L\}\}$, are feature vectors whose elements are functions of the target pixel’s color and intensity values. The size of each codebook, the number of codewords $|C|$, is adapted according to the specific appearance dynamics observed at each pixel location. A codeword $c_m \in C$ is a three-tuple $c_m = \{v_m, aux_m, \sigma^2_m\}$ where $v_m$ records the average color associated with the codeword, $aux_m$ encodes the brightness and temporal statistics of the codeword and $\sigma^2_m$ is the variance of a color.
distortion measure. Specifically, \( aux_m \) is defined as

\[
aux_m = (I_{min}^m, I_{max}^m, f_m, \lambda_m, q_{first}^m, q_{last}^m)
\]  

(2.27)

where \( I_{min}^m \) and \( I_{max}^m \) are the minimum and maximum brightness values of all observations associated with the codeword where brightness is defined as the L2 norm of the RGB color vector

\[
I = \sqrt{R^2 + G^2 + B^2}
\]

(2.28)

\( f_m \) is the number of observations that have been associated with a codeword, \( q_{first}^m \) and \( q_{last}^m \) are the first and last times the codeword occurred and \( \lambda_m \) is the Maximum Negative Run-Length (MNRL) defined as the longest period of time during training that a codeword did not occur.

Given a codeword \( c_l \) and an observation \( y_t \), a color distortion distance \( \delta(y_t, c_l) \) is proposed as a measure of codeword and observation color similarity

\[
\delta(y_t, c_l) = \frac{\sqrt{||y_t||^2 - p^2}}{\sigma_l}
\]

(2.29)
where

\[ ||y_t||^2 = \langle y_t, y_t \rangle = R^2 + G^2 + B^2 \]
\[ ||v_l||^2 = \langle v_l, v_l \rangle = \tilde{R}^2 + \tilde{G}^2 + \tilde{B}^2 \]
\[ \langle y_t, v_l \rangle^2 = R^2 \tilde{R}^2 + G^2 \tilde{G}^2 + B^2 \tilde{B}^2 \]
\[ \rho^2 = \frac{||y_t||^2 \cos^2 \theta = \frac{\langle y_t, v_l \rangle^2}{||v_l||^2}}{||v_l||^2} \]  

(2.30)

An observation is considered to match a particular codeword \( c_l \) if the intensity of the observation is bounded by the minimum and maximum intensity parameters of the codeword, \( I_l^{\text{min}} \leq I \leq I_l^{\text{max}} \) and the color distortion between the observation and the target codeword is less than a global tolerance: \( \delta (y_t, v_l) \leq \epsilon \). The brightness and color constraints define a cylindrical boundary in RGB colorspace visualized in figure 2.11. The centroid and orientation of the cylinder correspond to the endpoint and direction of the average color vector associated with the codeword, the length of the cylinder is proportional to the codeword brightness range and the color distance threshold defines the radius of the cylinder. Pixel color observations lying within or on the cylindrical codeword decision boundary are considered to be matching the codeword with matches ranked according to the color distortion measure.

The algorithm for constructing a codebook at a given pixel is outlined in listing 1. Initially, the codebook is empty and the initial observation is used to create the first codeword:

\[ c_1 = \{ v_1 = (R, G, B), aux_1 = (I, I, 0, 1, 1), \sigma_{\text{init}} \}. \]  

(2.31)

where the global parameter \( \sigma_{\text{init}} \) initializes the color distortion variance estimation. Subsequent observations at a given pixel location are compared to the existing set of codewords. If any codewords match the observation, the best match in terms of minimal color distortion, denoted \( c_{\text{min}} \), is updated according to:

\[ v_{\text{min}} \leftarrow \left( \frac{(1 - \gamma)f_m \tilde{R} + \gamma R}{f_m + 1}, \frac{(1 - \gamma)f_m \tilde{G} + \gamma G}{f_m + 1}, \frac{(1 - \gamma)f_m \tilde{B} + \gamma B}{f_m + 1} \right) \]
\[ aux_{\text{min}} \leftarrow \left( \min \left\{ I_{\text{low}}, \min \{ I, I_{\text{min}}^{\text{min}} \} \right\}, \max \left\{ I_{\text{hi}}, \max \{ I, I_{\text{min}}^{\text{max}} \} \right\}, f_m + 1, \max \left\{ \lambda, t - q_{\text{first}}^{\text{first}} \right\}, q_{\text{last}}, t \right) \]
\[ \sigma_{\text{min}}^2 \leftarrow \left( \rho \delta (y_t, c_{\text{min}})^2 + (1 - \rho) \sigma_{\text{min}}^2 \right) \]  

(2.32)
Figure 2.11: CBBGS codeword geometry. A per-pixel codeword defines a cylindrical decision boundary in the RGB color space. For an observation \( y_t \) to be considered a match to a particular codeword \( c_m \), the observation must lie inside the codeword’s cylindrical boundary. The color distance threshold \( \epsilon \) constrains the radius of the boundary while the brightness constraints controls the height. Matching observations adapt the position and orientation of the decision boundary. Image adapted from [6].

where \( I_{\text{low}} \) and \( I_{\text{hi}} \) bound the codeword brightness range and are functions of the global parameters \( \alpha \) and \( \beta \):

\[
I_{\text{low}} = \alpha I_{m}^{\text{min}} \quad I_{\text{hi}} = \beta I_{m}^{\text{max}}
\]  

(2.33)

With \( \alpha < 1 \) and typically set in the range \( 0.4 \leq \alpha \leq 0.7 \) and \( \beta > 1 \) with typical range \( 1.1 \geq \beta \geq 1.5 \).

If there are no matches, a new codeword is added to the codebook according to:

\[
v_{L+1} = (R, G, B) \quad aux_{L+1} = (I, I, t - 1, t, t)
\]  

(2.34)

A partial list consisting of the fundamental parameters of the CBBGS algorithm and the values suggested by [6] are presented in table 2.6.

CBBGS initially stores all code words created during training but during a second pass of the training data, any code word with a MNRL less than a pre-defined threshold denoted \( \lambda^{\text{thresh}} \) and set in the experiments reported in [6] as half the length of the training sequence, is considered foreground and discarded from the codebook. The resulting “pruned” codebook is the final compressed
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Suggested Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>Global color distortion threshold</td>
<td>Not Reported</td>
</tr>
<tr>
<td>$\sigma_{\text{init}}^2$</td>
<td>Initial codeword color distortion variance estimate</td>
<td>Not Reported</td>
</tr>
<tr>
<td>$\lambda_{\text{thresh}}$</td>
<td>MNRL threshold</td>
<td>Half the length of the training sequence.</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Scale factor for maximum brightness bound</td>
<td>$0.4 \leq \alpha \leq 0.7$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Scale factor for minimum brightness bound</td>
<td>$1.1 \geq \beta \geq 1.5$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>RGB average learning rate</td>
<td>Not Reported</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Color distortion variance learning rate</td>
<td>Not Reported</td>
</tr>
</tbody>
</table>

Table 2.6: Table of fundamental parameters for the CBBGS algorithm [6].

**Algorithm 1** CBBGS codebook construction algorithm adapted from [6]

1: $L \leftarrow 0$, $\mathcal{C} \leftarrow \emptyset$
2: for all $t \in \{1, \ldots, T\}$ do
3: $y_t = (R, G, B), I = \sqrt{R^2 + G^2 + B^2}$
4: $C_{\text{match}} \leftarrow \emptyset$
5: for all $c_m \in \mathcal{C} = \{c_i \mid 1 \leq i \leq L\}$ do
6: if $I_{\text{min}}^{m} \leq I \leq I_{\text{max}}^{m}$ \& $\lambda_{(y_t, v_m)} / \sigma_{m}^{2} < \epsilon$ then
7: $C_{\text{match}} \leftarrow \{c_m\} \cup C_{\text{match}}$
8: end if
9: end for
10: if $C_{\text{match}} \neq \emptyset$ then
11: $c_{\text{min}} = \arg \min_{c_i \in \mathcal{C}_{\text{match}}} \{\delta(x_t, y_t) \mid c_i \in \mathcal{C}_{\text{match}}\}$
12: $\sigma_{\text{min}} \leftarrow \rho_{\text{m}}/(y_t, c_{\text{min}})^2 - (1 - \rho)\sigma_{\text{min}}^{2}$
13: $v_{\text{min}} \leftarrow \frac{(1-\gamma)f_{\text{min}}R_{\text{min}} + \gamma R}{f_{\text{min}} + 1}, \frac{(1-\gamma)f_{\text{min}}G_{\text{min}} + \gamma G}{f_{\text{min}} + 1}, \frac{(1-\gamma)f_{\text{min}}B_{\text{min}} + \gamma B}{f_{\text{min}} + 1}$
14: $\text{aux}_{\text{min}} \leftarrow \min \{I, I_{\text{min}}^{\text{m}}, \max \{I, I_{\text{max}}^{\text{m}}, f_{\text{m}} + 1, \max \{\lambda_{m}, t - q_{\text{first}}^{\text{m}} \}, q_{\text{last}}^{\text{m}}, t\}\}$
15: else
16: Create a new codeword with $v_{L+1} \leftarrow (R, G, B)$ and $\text{aux}_{L+1} \leftarrow (I, I, 1, t - 1, t, t)$
17: $\mathcal{C} \leftarrow \{c_{L+1}\} \cup \mathcal{C}$
18: end if
19: end for
20: For each codeword $c_i \in \mathcal{C}$, wrap around $\lambda_i$ by setting $\lambda_i \leftarrow \max \{\lambda_i, (T - q_{\text{last}}^{\text{m}} + q_{\text{first}}^{\text{m}} - 1)\}$
background model. To detect change, every pixel observation is compared to the appropriate codebook and if there is a matching codeword, the observation is assumed to be a sample corresponding to normal scene dynamics. If there is no match the observation is considered to be associated with a significant change.

Kim et al. continue to define additional heuristics and background modeling layers for reasoning about changes using the codebook vocabulary. However, a fundamental weakness of the CBBGS algorithm is its sparse temporal model which combined with the MNRL threshold causes many of the same problems associated with ViBe and PBAS. While the codewords provide a compact representation of low level appearance compared to explicitly storing pixel intensities, the global MNRL threshold places an upper bound on the class of periodic phenomena that can be modeled by the CBBGS per-pixel code books.

2.7 Domain Transformation Pixel Models

Domain transformations are a general tool for representing a function over one domain by an equivalent representation in another space. In practice, the original function is approximated by a truncated, finite linear combination of weighted orthonormal functions. The weights of each orthonormal function are sufficient to represent the original signal in the alternative domain and are commonly referred to as coefficients or responses. Many different domain transformations have been applied to the change detection problem [62] including the Fast Fourier Transform (FFT) [8], and Discrete Cosine Transform (DCT) [7]. Wavelet transforms are another popular class of domain transformations that typically operate on local spatial neighborhoods as well as temporal extents and are therefore classified as region-class change detection methods [39, 40, 41, 42].

Wren and Porikli propose a series of extremely similar background modeling techniques based on the frequency decomposition of pixel intensity sequences using the Discrete Cosine Transform (DCT) in [7] and the Fast Fourier Transform in [8]. The algorithms proposed in [7] and [8] are
named Wave-Back and Waviz which we maintain for clarity.

Wren and Porikli argue that state of the art change detection systems model scenes using appearance information only, ignoring temporal information when building models for normal scene appearance, a concern substantiated by the literature review presented thus far. The SG-GMM and DP-GMM algorithms outlined in section 2.2 and 2.3 assumed that pixel intensities or colors were drawn independently from a static mixture of Gaussians, the per-pixel intensity model introduced by Elgammal et al. was likewise unable to account for the regularity of the blade appearance as highlighted in figure 2.8c, the nonparametric algorithms ViBe and PBAS do not explicitly model appearance dynamics and treat short-term periodic phenomena as an after-thought using post-processing to discover “blinking” pixels, and the codebook method introduced by Kim et al. proposed the MNRL; a single number to representing limited temporal information about codeword appearance durations and the underlying dynamical process driving normal appearance patterns. The Hidden Markov Models used by Leotta and Paragios are a first step forward towards relaxing the traditionally strict temporal independence assumptions traditionally imposed by change detection algorithms yet HMMs model only local temporal information and are incapable of explicitly accounting for appearance duration as explored in section 2.4 and shown by the implicit geometric duration distribution visualized in figure 2.5.

The danger of assuming pixel appearances are drawn independently from a static distribution is illustrated by Porikli and Wren in figure 2.12. Porikli correctly argues that two signals that are clearly distinct when temporal information is considered may lead to indistinguishable appearance models when observations are incorrectly assumed to be temporally independent.

To address these shortcomings, Wren and Portkili propose analyzing the frequency decomposition of per-pixel intensity sequences in video using a sliding window approach. Both Wave-Back and Waviz store a list of the last \( N - 1 \) intensity values observed at each pixel. When a new frame of video is observed, the new pixel intensity value is appended to the appropriate list resulting in an intensity sequence of length \( N \): \([y_0, y_1, \ldots, y_{N-1}]\) where the elements are zero indexed in keeping
with the conventions in [7] and [8]. Wave-Back uses the Discrete Cosine Transform defined by equation 2.35 and Waviz uses Discrete Fourier Transform of equation 2.36 to decompose the intensity sequence of the $N$ length intensity sequence into a list of $K$ frequency coefficients. When the next frame of video is ready for processing, the last element of the pixel intensity list is popped off the end and the newest pixel intensity is added to the beginning.

$$c_k = \sum_{n=0}^{N-1} y_n \cos \left( \frac{\pi n (2n + 1)}{2N} \right)$$  \hspace{1cm} (2.35)$$

$$c_k = \sum_{n=0}^{N-1} y_n e^{i 2 \pi k n N}$$  \hspace{1cm} (2.36)$$

In general, the DFT coefficients are complex numbers but the per-pixel intensity sequence model proposed by Waviz uses only the magnitude of each coefficient, $||c_k||$. To account for slowly changing scene dynamics, both algorithms maintain a background model composed of a vector which is the recursive accumulation of previously computed frequency coefficients weighted by the exponential forgetting parameter $\alpha$

$$\hat{c}_k \leftarrow \alpha \hat{c}_k + (1 - \alpha) c_k$$  \hspace{1cm} (2.37)$$

Wave-Back and Waviz use the L2 norm of coefficient vectors to measure the similarity between intensity sequences in the frequency domain assuming intensity sequences that are associated with
the same process will have a small coefficient distance and sequences generated by different processes will have a large coefficient distance. The distance between the accumulated coefficients from equation 2.37 and coefficients computed for the current observation sequence is therefore a measure of similarity between the current intensity sequence and the background model

\[ d = \sqrt{\sum_{k=1}^{N-1} (\hat{c}_k - c_k)^2} \]  

(2.38)

Because observations are accumulated incrementally, the instant a change begins, there will be approximately one intensity value corresponding to the change in the list of \( N \) accumulated values.

Depending on the size, velocity and direction of the abnormality, the ratio of the number of intensity values associated with the change and the number of intensity values associated with background will increase at a particular rate. At some point, the real-world dynamic associated with the appearance changes will stop and the background appearances will return. The number of intensity values stored in history that are associated with the appearance of the change will decay relative to the number of intensity values corresponding to the background process. This leads to a gradual increase and decrease in the coefficient-background distance measure and casts a false tail of large coefficient distance preceding and following an abnormal event as visualized in figure 2.13b. The brightness of each pixel in figure 2.13b is proportional to the distance defined in equation 2.38.

To compute the frequency coefficients accurately, a large history of past observations is preferable yet accumulating a larger history of intensities results in more temporal ambiguity as to which observations are responsible for the resulting set of coefficients. To remove the “distance trail”, Wave-Back stores a record of the \( M - 1 \) previously computed coefficient-background distances along with the current coefficient distance resulting in a list of the \( M \) most recent distance: \([d_0, d_1, \ldots, d_M]\) and computes the product of distances in equation 2.39 in attempt to remove the trail artifacts.

\[ \hat{d} = \prod_{m=0}^{M} d_m \]  

(2.39)

The “trail” and the efficacy of the modified distance is shown in figure 2.13 which originally appeared in [7]. Figure 2.13a shows a frame from a traffic camera where the DCT background coefficients
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Reported Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>16</td>
</tr>
<tr>
<td>N</td>
<td>16</td>
</tr>
<tr>
<td>M</td>
<td>16</td>
</tr>
<tr>
<td>α</td>
<td>Not Reported</td>
</tr>
</tbody>
</table>

Table 2.7: Parameters of Wave-Back and experimental values reported in [7].

computed by the Wave-Back algorithm have been trained on the road segment. Figure 2.13b visualizes the original distance measure computed according to equation 2.38 and figure 2.13c with bright pixels corresponding to a large distance. The effect of modifying the distance by the running product of recent distances is visualized in figure 2.13c.

![Frame of traffic video.](image1) ![Coefficient distance map with trail.](image2) ![Distance map after tail removal by equation 2.37.](image3)

Figure 2.13: Distance trail and removal adapted from [7].

Wave-Back uses the modified distance in equation 2.39 while Waviz uses the coefficient distance from equation 2.38 without trail removal. Both algorithms define change as a value of $\hat{d}$ or $d$ that is greater than a threshold. Wave-Back requires $O(N + M + K)$ storage to maintain the list of $N$ historical intensity values, $M$ previous distance measures and $K$ background coefficients, and Waviz $O(N + K)$, similar memory requirements as the nonparametric algorithms in section 2.5. A simple implementation of both the DCT and DFT requires $O(NK)$ time to decompose a single intensity sequence. A complete list of parameters and their suggested values according to [7] and [8] are listed in tables 2.7 and 2.8 respectively.

Like ViBe and PBAS, Wave-Back and Waviz maintain a fixed sized history of intensity values. However, Wave-Back and Waviz store an ordered list of previous intensity observations whereas
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Reported Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>32, 64, 128</td>
</tr>
<tr>
<td>N</td>
<td>32, 64, 128</td>
</tr>
<tr>
<td>α</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 2.8: Waviz parameters and experimental values reported in [8]. In all experiments $K = N$ and experiments were conducted with $N = 32, 64, 128$ where it was determined empirically that $N = K = 32$ gave the best results.

ViBe and PBAS maintain an unordered collection whose elements randomly replaced according to a set of heuristics. Wave-Back and Waviz both suffer from practical limitations in that the length of the per-pixel sliding window places an upper limit on the temporal scale of patterns the background model can capture. Additionally, increasing the size of the window size to capture low frequency content sacrifices temporal resolution, making it difficult to determine exactly when a significant change began and ended.
Chapter 3

Modeling intensity sequences for pixel-level change detection

Coherent motion captured by surveillance cameras typically induce predictable spatio-temporal appearance sequences in video, yet the state-of-the-art change detection algorithms explored in chapter 2 attempt to model common pixel-level appearance values rather than sequences of appearances. For example, while the SG-GMM, DP-GMM, GKDE, ViBe, PBAS, and CBBGS algorithms utilize dynamic learning schemes to adjust model parameters over time, for a fixed set of parameters each model assumes per-pixel observations are a collection of exchangeable random variables; the probability of a series of observations is assumed to be invariant with respect to permutation. It could be argued that the domain transformations algorithms, Wave-Back and Waviz and HMM based algorithms model intensity sequences rather than intensity values. However, the HMM is merely a first order relaxation of strict temporal independence assumptions imposed by the GMM. As discussed in section 2.4, as a consequence of the local state transition model and implicit geometric state duration distribution, the HMM can accurately model only a limited set of stochastic processes. While the frequency decomposition of intensity sequences has the potential to model temporal features
present in the per-pixel signal, the implementations of the decompositions result in time-frequency resolution tradeoffs which limit the performance of Wave-Back and Waviz.

This chapter describes a supervised pixel-level change detection algorithm for fixed, monocular surveillance cameras that directly addresses the lack of temporal context in state-of-the-art per-pixel change detection methods. A per-pixel intensity sequence model based on a sub-class of hidden semi-Markov models (HSMM) duration dependent hidden Markov models (DDHMMs), is introduced in section 3.1. Data structures and algorithms for the dynamic quantization of time are developed in 3.2 and are used as representation of latent state duration distributions that captures the appropriate time scale of state conditional durations while adapting the temporal resolution in areas of high transition probability. Given example intensity sequences at each pixel location, the parameters of each independent DDHMM are learned by a sequential estimation algorithm detailed in section 3.3. The parameter estimation algorithm simultaneously assigns discrete state labels to pixel intensity sequences that summarize the appearance and temporal statistics of a subsequence of observations. Section 3.7 explains how state assignments are then used as features for constructing pixel-level code books during a training phase to identify changes of interest in new video.

The per-pixel model is validated by showing superior intensity sequence predictive performance to pixel representations commonly used in change detection applications. A new data set is presented which contain dynamic periodic backgrounds with larger time scale variability than previously exhibited in standard change detection benchmarks and the proposed method is compared to state-of-the-art change detection methods using the new videos. Tables 3.1 and 3.2 provide a concise reference for the notation and parameters introduced and used throughout this section.
Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>Intensity observation</td>
</tr>
<tr>
<td>$Y$</td>
<td>Sequence of intensity observations: $Y = {y_1, \ldots, y_T}$</td>
</tr>
<tr>
<td>$t$</td>
<td>Index of observations</td>
</tr>
<tr>
<td>$T$</td>
<td>Total number of observations</td>
</tr>
<tr>
<td>$K$</td>
<td>Integer number of latent states</td>
</tr>
<tr>
<td>$M^K$</td>
<td>$DDHMM$ model with $K$ states</td>
</tr>
<tr>
<td>$M^{K+1}$</td>
<td>Identical copy of $DDHMM$ but extended to include an extra state $s_{K+1}$</td>
</tr>
<tr>
<td>$l$</td>
<td>$DDHMM$ segment index</td>
</tr>
<tr>
<td>$L$</td>
<td>Total number of $DDHMM$ segments, a random variable</td>
</tr>
<tr>
<td>$S_l$</td>
<td>State label random variable $S_l \in {s_k \mid k \in {1, \ldots, K}}$</td>
</tr>
<tr>
<td>$s_l$</td>
<td>State label assigned to $DDHMM$ segment $l$</td>
</tr>
<tr>
<td>$q_t$</td>
<td>State label assigned to observation at time $t$</td>
</tr>
<tr>
<td>$Q$</td>
<td>Sequence of state assignments in one-to-one correspondence with observations $Q = {q_t \mid t \in {1, \ldots, T}}$</td>
</tr>
<tr>
<td>$s^c$</td>
<td>Current state at a given step during training $s^c \in {s_k \mid k \in {1, \ldots, K}}$</td>
</tr>
<tr>
<td>$s^c$</td>
<td>Best local state assignment associated with observation $y_t$ from the collection of $K$ existing states</td>
</tr>
<tr>
<td>$D_l$</td>
<td>Duration random variable for segment $l$</td>
</tr>
<tr>
<td>$[d_{\text{min}}, d_{\text{max}}]$</td>
<td>Quantized finite interval of allowable state durations</td>
</tr>
<tr>
<td>$</td>
<td>D</td>
</tr>
<tr>
<td>$r_l$</td>
<td>Time index of the beginning of segment $l$: $r = \sum_{i=1}^{l-1} D_l$</td>
</tr>
<tr>
<td>$\theta_k$</td>
<td>State emission parameters associated with state $s_k$</td>
</tr>
<tr>
<td>$\Theta$</td>
<td>Collection of all state conditional emission parameters $\Theta = {\theta_k \mid k \in {1, \ldots, K}}$</td>
</tr>
<tr>
<td>$\theta^c$</td>
<td>State emission parameters of the current state</td>
</tr>
<tr>
<td>$s_{K+1}$</td>
<td>Proposed new state</td>
</tr>
<tr>
<td>$\mu_k$</td>
<td>Mean of emission density associated with state $s_k$</td>
</tr>
<tr>
<td>$\sigma_k$</td>
<td>Standard of emission density associated with state $s_k$</td>
</tr>
<tr>
<td>$n_k$</td>
<td>Number of observations associated with state $s_k$</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of notation used for Duration Dependent Markov Model.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\text{init}}$</td>
<td>Global prior state standard deviation</td>
<td>15.0</td>
</tr>
<tr>
<td>$\sigma_{\text{min}}$</td>
<td>Global minimum state standard deviation</td>
<td>3.0</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Global multinomial smoothing parameter</td>
<td>1.0</td>
</tr>
<tr>
<td>$\Delta_{\text{min}}$</td>
<td>Global minimum temporal resolution of component binary trees at root</td>
<td>4.0</td>
</tr>
<tr>
<td>$\Delta_{\text{max}}$</td>
<td>Global maximum temporal resolution of component binary tree at leaves</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 3.2: $DDHMM$ learning algorithm parameters and default values. If a parameter is labeled “Global”, then the single parameter applies to models at each independent pixel location.
3.1 Per-Pixel Intensity Sequence Model

![Diagram of HSMM](image)

Figure 3.1: Visualizing the general HSMM as a probabilistic graphical model. A sequence of observations $Y = \{y_1, \ldots, y_T\}$ of length $T$ is broken up into $L$ segments or sub-sequences that are related to the latent state-duration pairs through a conditional emission density. Note, $L$ is a random variable but is omitted from the illustration for clarity.

Figure 3.2: A concrete example of an HSMM. The duration random variable at each segment is conditioned on the duration of the previous segment and the current state. The state transition distribution, in this example, is dependent on the previous state and the previous duration. The likelihood of each observation sub-sequence is dependent on the duration random variable but the links are omitted for clarity.

A Hidden Semi-Markov Model is a probabilistic discrete latent state space model similar to the GMM and HMM except both state transitions and durations are explicitly modeled. HSMMs refer to a general class of models that encapsulate various factorizations of the conditional independence
Figure 3.3: Visualizing the temporal independence assumptions of Duration Dependent Markov Model. The duration distribution is conditioned only on the current state and the observations sequence is modeled as a collection of independent identically distributed (i.i.d.) random variables with size equal to the value of the segment duration. Plate notation is used to concisely represent the i.i.d. emission density assumption given a state-duration pair.

assumptions imposed on the conditional duration, transition and emission densities. In general, a HSMM models a sequence of observations $Y = \{y_1, \ldots, y_T\}$ using a sequence of latent state pairs: $((S_1, D_1), (S_2, D_2), \ldots, (S_L, D_L))$ where $S_i$ is a state label and $D_i$ is a random variable that represents the time spent in state $S_i$. The state-duration tuples partition the complete observation sequence into an ordered set of $L$ segments or sub-sequences where $L$ is a random variable. The probabilistic graphical model for a HSMM is shown in figure 3.1 where the node representing the number of segments $L$ is omitted for clarity. Note the variable topology of the graph in figure 3.1; the number of nodes and conditional relationships will change depending on the given value of $L$ and the configuration of the duration random variables. There are many ways to visualize a HSMM, some include incorporating auxiliary random variables that represent the time sub-sequences stop and start to establish a fixed graphical topology [17, 19, 18], an example of which is shown in figure 3.4. However, the representation in figure 3.1 most clearly establishes the fundamental quantities and conditional relations assumed by the HSMM without polluting the illustration with additional variables.
Figure 3.4: Alternative DDHMM representation using counter and finishing random variables. To establish a graphical model for the DDHMM with a fixed topology, rather than directly modeling duration, counter and finishing random variables are introduced. There is a state indicator random variable \( \{S_1, \ldots, S_T\} \) for each observation, \( \{y_1, \ldots, y_T\} \). The counter random variables \( \{C_1, \ldots, C_T\} \) represent the remaining time left in given homogeneous state segment while the finishing random variables \( \{F_1, \ldots, F_T\} \) takes the value of one only when there is a state transition \( S_t \neq S_{t-1} \) and zero elsewhere. This figure was influenced by similar graphical models appearing in [17, 18] and [19].

An example of a concrete HSMM with a particular set of conditional independence assumptions is visualized in figure 3.2. In the most general case, as visualized in 3.1, the probability of a state-duration tuple at a given segment is jointly dependent on the previous segment

\[
p(S_l, D_l \mid S_{l-1}, D_{l-1})
\]

(3.1)

However, the graphical model in figure 3.2 specifies the following factorization:

\[
p(S_l, D_l \mid S_{l-1}, D_{l-1}) = p(S_l \mid D_{l-1}, S_{l-1}) p(D_l \mid S_l, D_{l-1})
\]

(3.2)

Assuming the state duration random variable can take a value on a bounded finite interval \([d^{\text{min}}, d^{\text{max}}]\) discretized with \(|D|\) unique values, using a conditional probability table to represent the distribution of state-duration pairs conditioned on the previous state-duration pair would require a table of size \(\mathcal{O}\left((K \cdot |D|)^{K \cdot |D|}\right)\) where \(K\) is the total number of states. Under the same assumptions, factoring the conditional state-duration distribution according to equation 3.2 results in a table of size \(\mathcal{O}\left(K^{K \cdot |D|} + |D|^{K \cdot |D|}\right)\). These examples illustrate how specific probabilistic factorizations modify the storage and processing complexity of the HSMM.

In the context of this research, a Duration Dependent Hidden Markov Model (DDHMM) refers
to a HSMM with the following joint state-duration distribution factorization

$$p(S_t, D_t \mid S_{t-1}, D_{t-1}) = p(S_t \mid S_{t-1}) p(D_t \mid S_t)$$  \hspace{1cm} (3.3)$$

which may be fully specified by a conditional probability table with $O(K^2 + K |S|)$ numbers. Without loss of generality, it may assumed that there are no self-transitions between segments, $s_t \neq s_{t-1}$, reducing the representation by $O(|S|)$. Modeling a discrete time sequence of length $T$ using a DDHMM assumes the underlying physical phenomena may be accurately approximated by the following stochastic process:

**Generative Model 4. DDHMM**

1. Choose an initial state $s_1 \sim p(S)$

2. Choose a duration $d_1 \sim p(D \mid s_1)$

3. Sample a sequence of observations $\{y_1, \ldots, y_{d_1} \mid y_i \sim p(y \mid s_1)\}$

4. Set a counter $r = d_1$

5. while $r \leq T$

   (a) Choose a state $s_l \sim p(S_l \mid S_{l-1})$

   (b) Choose a duration $d_l \sim p(D_l \mid S_l)$

   (c) Sample a sequence of observations $\{y_r, \ldots, y_{r+d_l} \mid y_i \sim p(y \mid s_l)\}$

   (d) $r = r + d_l$

Figure 3.3 visualizes the DDHMM as a probabilistic graphical model. Like the general HSMM, the topology of the DDHMM graphical model is variable as the number of state-duration tuples will change depending on the particular configuration of the duration random variables.

The observation and state sequences are related through three fundamental distributions: the duration $p(D_t \mid S_t)$, state transition $p(S_t \mid S_{t-1})$ and emission $p(y_t \mid S_t)$ distributions. The joint
probability of the observation and latent state-duration sequences is

\[
p(y_1, \ldots, y_T, S_1, D_1, \ldots, S_L, D_L) = p(S_1) p(D_1 \mid S_1) \prod_{l=1}^{D_1} p(y_i \mid S_1) \cdots
\]

Initial state-duration

\[
\prod_{l=2}^{L-1} p(D_l \mid S_l)p(S_l \mid S_{l-1}) \prod_{j=1}^{D_l} p(y_{r+j} \mid S_l)
\]

Internal state-durations, transitions and emission

\[
p(D \geq D_L \mid S_L)p(S_L \mid S_{(L-1)}) \prod_{z=1}^{D_L} p(y_{r(L-1)+z} \mid S_L)
\]

right-censor term

(3.4)

Where \( r_i = \sum_{m=1}^{l} D_m \) and \( p(S_1) \) is an initial distribution of state labels. The observation sequence is assumed to be right-censored, i.e., the last tuple \((S_l, D_l)\) is distributed according to the state survival distribution \( p(D \geq D_l \mid S_l) \), to mitigate the effect of the length of the observation sequence on the probability of a particular state sequence [63].

In all experiments the duration and transition distributions are multinomial where self-transitions between segments, \( S_l = S_{l-1} \), are forbidden as outlined in section 3.2. The emission distributions conditioned on \( S_l = s_k \) are univariate Gaussian with mean and standard deviations \( \{\mu_k, \sigma_k\} \).

### 3.2 Adaptive Duration Distribution & Epoch Granularity

The duration distribution of a DDHMM is commonly restricted to a parametric family, such as the Poission, Gamma, or Rayleigh densities, minimize storage and processing [64, 18, 65]. However, similar to [66], in this work the duration distribution \( p(D \mid S) \) is modeled using a histogram. To address storage and processing efficiency, the proposed histogram representation adapts the temporal quantization to the natural scale of the observed phenomena and refines the granularity in areas of high transition probability.

Temporal scale is a phenomena humans deal with on a subconscious level every day. For example, when measuring the time between eruptions of the Old Faithful geyser in Yellow Stone National Park, a common estimate would be 70 minutes. However, other equally correct answers include 4200 seconds, 4200000 milliseconds, 1.67 hours or 0.048 days. Yet, qualitatively 70 minutes seems
the most natural answer. In the context of the proposed DDHMM model, selecting a particular time scale for quantizing the duration distribution \( p(D \mid S) \) will have a significant effect on the predictive performance of the model and the run-time complexity for parameter estimation algorithms. For example, if the duration distribution is represented using a histogram over a particular duration range, what quantization should be used? It is probably inefficient to quantize duration at the frame rate as coherent actions typically induce homogeneous intensity values that persist for a series of frames. However, a coarse quantization will induce a smoothing bias, loss of temporal resolution, and border effects as it is not clear where digital intervals should start and end. Similar to the Old Faithful example, the goal is to quantize the time line as efficiently as possible, allowing the system to capture subtle temporal variations while using as little resources as possible.

Figure 3.5: Visualizing the theoretically absolute time line in terms of relative time between events. Each state conditional duration distribution of the DDHMM measures the relative time between two events, when a particular state \( S_l = s_k \) begins, and when the system transitions to any other state, \( S_{l+1} \) with \( S_{l+1} \neq s_k \). At the correct time scale, only a single event can happen per temporal quantum.
Time is not absolute and can only be defined in terms of the duration between events. For physicists, time is measured relative to the big bang, UNIX operating systems define time in terms of the number of seconds that have elapsed since Thursday, 1 January 1970 (this is also known as POSIX or Epoch time), and atomic clocks like the one shown in figure 3.5a, utilize the rate at which atoms emit radiation where the most commonly used element in atomic clocks, Caesium, is visualized in 3.5b. In order to determine an appropriate scale for quantizing latent state duration, it is necessary to establish state transitions as relative epochs and attempt to determine the natural scale at which epochs occur. To accurately and efficiently model duration, the following principles should be followed:

**Principle 1.** *Two events cannot occur at the same time.*

**Principle 2.** *Do not use a finer time scale than necessary.*

In terms of estimating the duration distribution for the DDHMM, when a given state $S_l = s_k$ is associated with a group of observations, the only event of interest is the transition to any other state $S_{l+1} \neq s_k$ at some time relative to the beginning of $S_l$. For a given state, transitioning to any other state after persisting for 12 time steps is a different event than transitioning to any other state after sojourn of 15 time steps. Principle 1, that no two events occur at the same time, maintains that these two different events should be recorded as unique, not obscured and considered as a single event. This requires a temporal quantization that is fine enough to associate the events with distinct quanta. A trivial way to satisfy principle 1 would be to use the finest temporal quantization granularity possible. However, in most cases this will quickly and needlessly exhaust resources. Principle 2 provides a regularization of principle 1 expressing the desire to use the largest time scale possible while preserving the ability to disambiguate separate events.

To satisfy principles 1 and 2, each duration distribution is modeled using a forest of fixed depth binary trees. The temporal extent of each binary tree component is bounded by the tree depth parameters $\Delta_{\text{min}}$ and $\Delta_{\text{max}}$ that express the minimum and maximum temporal resolution of the
tree as a power of two. For example a value of $\Delta^{\text{min}} = 4$ means the minimum temporal resolution of the component tree, the root of the tree, spans $2^4 = 8$ time units (frames in all experiments) and $\Delta^{\text{max}} = 0$ means the temporal resolution of a terminal leaf is $2^0 = 1$, a single time unit. Figure 3.6 visualizes a duration timeline quantized using binary trees. Figure 3.7 visualizes a the process of adaptively quantizing a segment of time using the single fixed depth binary tree to the left and the segment of time spanned at each level of the tree to the right.

At initialization, each duration histogram contains only the forest root node. When a duration value is added to the histogram, binary trees are added to the forest until the temporal extent of the forest is greater than the value to be added. The component which spans the temporal range that includes the value to be added is traversed to the nearest leaf. If the depth of the leaf is lower than $\Delta^{\text{max}}$, two children are added to the leaf, further subdividing the temporal in half. A single count is added to each node from the lowest leaf back to the root of the forest. In this way the count associated with the forest root is the total number of values added to the histogram. The root of each component is the number of values added to the components temporal range, and so forth down to the leaves of each tree. This process is continued for all subsequent values that are added to the histogram.
Figure 3.7: Partitioning a segment of the time line with a single binary tree component. (a) shows the relationship between the depth parameters $\Delta_{\text{min}}$ and $\Delta_{\text{max}}$ with corresponding temporal segments in (b).

Figure 3.8 is an example of the proposed adaptive histogram for a forest with depth parameters $\Delta_{\text{min}} = 4, \Delta_{\text{max}} = 0$ after observing the sequence $\{19, 18, 17, 18\}$. First, three component trees are added to the forest to span the initial value of 19. The right-most component tree is refined by the subsequent values which result in the adaptive quantization of time visualized by the red intervals.

Although a single binary tree may have been used to adaptively quantize the duration time line, a longer natural time scale of the natural phenomena would lead to a greater tree depth between the lowest temporal resolution at the root and the highest at the leaf nodes. The collection of binary trees approach limits the number of refinements necessary to reach the maximum temporal resolution through the tree depth parameters yet remains efficient to store and process.
Figure 3.8: Numerical example of the proposed adaptive histogram with $\Delta_{\text{min}} = 4$, $\Delta_{\text{max}} = 0$ after observing the sequence $\{19, 18, 17, 18\}$. The first value, 19 creates three trees to span the range 1-24, the values 18, 17, 18 further refine the third binary tree resulting in the quantization of the time line by the red bins with adaptive widths.

### 3.3 Learning DDHMM parameters

An incremental algorithm originally published in [32] is used for estimating the DDHMM model parameters and complexity. The learning algorithm defines three additional parameters: $\sigma_{\text{init}}, \sigma_{\text{min}}, \gamma$: the initial and minimum standard deviations for the state conditional emission densities, and multinomial smoothing parameter to avoid zero probabilities. Unless otherwise stated, $\sigma_{\text{init}} = 15.0$, $\sigma_{\text{min}} = 3.0$, and $\gamma = 1.0$ as summarized in table 3.2 which lists the DDHMM parameters proposed in this chapter along with default values.

For convenience, define $Q = \{q_1, \ldots, q_T\}$, a list of state indicator variables that map from DDHMM latent state segments to the observation time scale. For example, if observations $y_5, \ldots, y_9$ are associated with $(S_l = s_k, D = 5)$ then $q_5, \ldots, q_9$ are all equal to $s_k$, regardless of the segment index $l$. Given the first intensity observation $y_1$, a single state, $q_1 = s_1$ is created with parameters $\theta_1 = \{\mu_1 = y_1, \sigma_1 = \sigma_{\text{init}}, n_1 = 1\}$ where $n_k$ records the number of observations that have been associated with the $k^{th}$ state. The current state label $q_1$ is recorded, and a duration counter is initialized to $d = 1$. For all subsequent observations, the learning algorithm chooses one of three options: To extend the current temporal segment by associating a new observation with the current
state label, to initialize a new temporal segment by associating the new observation with a previously observed state label, or to initialize a new temporal segment that is associated with a new state which is added to the model in order to account for a previously unobserved appearance or temporal dynamic.

The best local state label assignment is made by choosing a state label from the set existing states according to

$$q^e = \arg \max_q \{p(y_t, q, q_{t-1}, d)\}$$

with

$$p(y_t, q, q_{t-1}, d) = \begin{cases} p(y_t \mid q)p(q \mid q_{t-1})p(d \mid q_{t-1}) & \text{if, } q \neq q_{t-1} \\ p(y_t \mid q)p(D \geq d \mid q_{t-1}) & \text{else.} \end{cases}$$

In Eq. 3.6, $p(y_t \mid q)p(q \mid q_{t-1})p(D = d \mid q_{t-1})$ is the probability of transitioning to a new state and $p(y_t \mid q_t)p(D \geq d \mid q_{t-1})$ is the probability of staying in the same state given the previous state assignment $q_{t-1}$ and intensity observation $y_t$.

The number of states for a given DDHMM at any point during learning is represented by the integer $K$. The probability of extending the current segment or transitioning to a previously existing state are additionally compared to the regularized likelihood of adding a new state to the current model, $s_{K+1}$, with an emission distribution centered around the intensity observed at time $t$, $y_t$, and initial standard deviation $\sigma_{\text{init}}$. The probability of transitioning to the newly proposed state is

$$p(s_{K+1} \mid s_{t-1}) = \frac{\gamma}{K + 1 + \sum_{s' \neq s_{t-1}} \#(s_{t-1} \rightarrow s' \mid d)}$$

where $\sum_{s' \neq s_{t-1}} \#(s_{t-1} \rightarrow s' \mid d)$ is the number of previously observed non-self transitions from the state associated with the previous segment $s_{t-1}$ to any other state after duration $d$. This is equivalent to having extended the multinomial transition distribution to have included a $s_{K+1}$ with $\gamma$ prior counts.

The algorithm then chooses between two DDHMMs, the DDHMM with the existing set of states $K$, and an extended DDHMM that is a copy of the current model but includes the hypothesized state...
The two models are denoted $M^K$ and $M^{K+1}$ respectively. To encourage a sparse allocation of resources and avoid over-fitting, the cost of making the decision to select the DDHMM with the extra state is regularized to favor simpler models, i.e., DDHMMs with fewer states, using the Akaike Information Criterion (AIC) [67].

The AIC score is a common regularization function used for such model selection. The AIC is defined as

$$AIC = 2\kappa - 2 \ln L(Y, \Theta)$$  \hspace{1cm} (3.8)

$\kappa$ is a measure of model complexity and $\ln L(Y, \Theta)$ is the maximal log-likelihood of an observation sequence with respect to model parameters $\Theta$. Using a duration histogram over a finite time interval quantized with $|D|$ values, multinomial transition and Gaussian emission distributions, and ignoring the compression of the duration histogram provided by the adaptive representation proposed in section 3.2, the complexity of DDHMM with $K$ states could be computed as

$$\kappa = K^2 - K + K|D| + 2K$$  \hspace{1cm} (3.9)

However, the transition histograms are dynamically allocated, extended as needed while the model is updated, and are typically very sparse. Therefore, $\kappa$ is approximated using the dominant terms in equation 3.9 as

$$\kappa = K^2 + K$$  \hspace{1cm} (3.10)

The the AIC score of the current DDHMM, $AIC^K$, and the new candidate model (the current model extended by adding $s_{K+1}$), $AIC^{K+1}$, as:

$$AIC^K = (K^2 + K) \ln [p(y_t, q^e, q_{t-1}, d)]$$

$$AIC^{K+1} = ((K + 1)^2 + K + 1) \ln [p(y_t | q_t = s_{K+1}) p(q_t = s_{K+1} | q_{t-1}) p(D = d | q_{t-1})]$$  \hspace{1cm} (3.11)

If $AIC^{K+1} < AIC^K$, $M^{K+1}$ is selected, meaning $s_{K+1}$ is added to $M^K$ and $s_{K+1}$ becomes the current state, otherwise $M^K$ is updated to reflect the maximal local state assignment over the $K$
existing states $s^e$. Regardless if the current or extended model is selected, the multinomial distributions are updated by adding a single count to the appropriate state conditional duration histogram and if $q_t \neq q_{t-1}$, the duration counter is reset to one, otherwise it is incremented. The parameters of the emission distribution for state $q_t = s_k$ are updated via standard sequential maximum likelihood using $\sigma_{\text{min}}$ as a minimum standard deviation for each appearance distribution [45].

The proposed DDHMM update and state assignment algorithm is detailed by the pseudocode in listings 4 and 2. The proposed learning algorithm, in conjunction with the efficient duration histogram representation described in section 3.2, facilitates an unoptimized multithreaded C++ implementation running on a 3.46 GHz Intel i7 processor to achieve real-time performance. Specifically, continuously updating a DDHMM at each pixel for a video sequence containing seventeen hundred frames with resolution $240 \times 320$ pixels takes an average of 31 milliseconds per frame.

**Algorithm 2** Local computation of the most likely state.

```plaintext
function \textit{argmax}_\textit{existing state}(y_t, \mathcal{M}, q, d)

1: \quad p_{\text{max}} \leftarrow 0
2: \quad q^e \leftarrow \emptyset
3: \quad \text{for all } s_k \in \mathcal{M} \text{ do}
4: \quad \quad \text{prob} \leftarrow 0
5: \quad \quad \text{if } s_k = q \text{ then}
6: \quad \quad \quad \text{prob} \leftarrow p(y_t \mid q)p(D \geq d \mid q)
7: \quad \quad \text{else}
8: \quad \quad \quad \text{prob} \leftarrow p(y_t \mid s_k)p(s_k \mid q)p(D = d \mid q)
9: \quad \quad \text{end if}
10: \quad \quad \text{if } \text{prob} > p_{\text{max}} \text{ then}
11: \quad \quad \quad p_{\text{max}} \leftarrow \text{prob}
12: \quad \quad \quad q^e \leftarrow s_k
13: \quad \quad \text{end if}
14: \quad \text{end for}
15: \quad \text{return } q^e
16: \end function
```
Algorithm 3 Sequential Gaussian Maximum Likelihood Update

1: function GAUSSIAN_UPDATE($y_t, M, s_k, \sigma_{min}$)
2: $n \leftarrow n_k + 1$
3: $\mu \leftarrow \mu_k + \frac{1}{n_k} (y_t - \mu_k)$
4: $\sigma \leftarrow \frac{(n_k-1)\sigma_k + (y_t - \mu_k)^2}{n_k}$
5: if $\sigma \leq \sigma_{min}$ then
6: $\sigma \leftarrow \sigma_{min}$
7: end if
8: $\theta_k \leftarrow \{\mu, \sigma, n\}$
9: end function

Algorithm 4 Per-Pixel DDHMM sequential learning algorithm

1: function DDHMM_UPDATE($y_t, M^K, q_{t-1}, d, \sigma_{init}, \sigma_{min}$)
2: $q_t \leftarrow \text{ARGMAX\_EXISTING\_STATE}(y_t, M^K, q_{t-1}, d)$
3: Create $s_{K+1}$ with associated parameters $\theta_{K+1} = \{\mu_{K+1} = y_t, \sigma_{K+1} = \sigma_{init}, n = 1\}$
4: $AIC^K \leftarrow (K^2 + K) \ln [p(y_t, q_t, q_{t-1}, d)]$
5: $AIC^{K+1} \leftarrow ((K + 1)^2 + K + 1) \ln (p(y_t | s_{K+1})p(s_{K+1} | q_{t-1})p(D = d | q_{t-1})$
6: if $AIC^{K+1} \leq AIC^K$ then
7: $M^K \leftarrow \{M^K \cup s_{K+1}\}$
8: $q_t \leftarrow s_{K+1}$
9: $d \leftarrow 1$
10: else
11: if $q_t = q_{t-1}$ then
12: $d \leftarrow d + 1$
13: else
14: $d \leftarrow 1$
15: end if
16: end if
17: if $q_t \neq s_{K+1}$ then
18: GAUSSIAN_UPDATE($y_t, M, q_t, \sigma_{min}$)
19: end if
20: end function
3.4 Qualitative Sample Analysis

This section qualitatively compares the proposed DDHMM and learning algorithm to the GMM and HMM for modeling for pixel intensity sequences. An intensity sequence is extracted at a given pixel location from four videos shown as the center of the highlighted circle in figures 3.9, 3.10, 3.11, 3.12, 3.13, (a) through (c), which are referred to as the Swing, Highway, Fields Point, Tehachapi and Helix videos respectively. Subfigure (d) in each figure plots the intensity as a function of time as observed at the target pixel. Subfigure (e) shows samples drawn from a Gaussian Mixture model with a varying number of states, $K = \{1, \ldots, 8\}$ from top to bottom. Subfigure (d) shows samples drawn from a Hidden Markov Model with Gaussian emission densities where each row again represents an HMM trained given a different number of states, $K = \{2, \ldots, 9\}$, from top to bottom. The final subfigures are samples taken from a DDHMM model trained using the intensity sequence and proposed learning algorithm from each video.

While there are many learning algorithms to choose from for estimating the parameters of GMM and HMM intensity models, including those reviewed in chapter 2, in each experiment, all GMMs and HMMs are trained using the Expectation Maximization (EM) algorithm. The EM algorithm is an iterative local parameter search which is guaranteed to increase the model likelihood with every iteration and converge to a parameter setting which is local optimum of the likelihood. The EM was selected rather than alternatives to emphasize that while the proposed DDHMM learning algorithm is a single-pass local search, it still outperforms the GMM and HMM when trained using the most theoretically powerful and most commonly applied learning algorithm from the literature. While the EM algorithm is a well-known general framework for maximum likelihood parameter estimation of latent variable models, a brief review deriving the fundamental equations for the specific cases of the GMM and HMMs is presented in appendix A.

The Swing sequence in figure 3.9 shows a mother pushing her daughter on a swing set. The periodic motion of the child against the background creates regular transitions between light and
dark intensity values as seen in the intensity sequence sample in figure 3.9d. Figures 3.9e and 3.9f visualizes samples drawn from GMM and HMM models trained using the EM algorithm for various numbers of states. Both the GMM and HMM models lack the temporal representation to accurately account for the regular intensity patterns present in the training sequence. The GMM does not model latent state temporal dynamics and therefore cannot reflect the periodic motion of the swinging child. While the HMM offers a first order temporal approximation, the model is incapable of capturing the dynamics of the original intensity sequence. However, the sampled sequences from the DDHMM closely mimic the original sequence and mirror the longer term ground - short term child appearance oscillation.

The highway sequence is more complex than the Swing video in that while cars exit and enter the scene at semi-regular intervals, each car has a different appearance. In the Swing video, the swinging child maintained a relatively constant appearance creating a regular pattern. The intensity sequences sampled from the GMM, as seen in figure 3.10e are dominated by intensity values close to the road with random bursts of other intensities corresponding to intensities of observed cars. While it is difficult to say if the HMM or DDHMM intensity sequence samples more closely reflects the training sequence, it is clear that at the very least the DDHMM does no worse than the HMM at capturing stretches of road intensities interspersed with intensities sampled from states that represent observed cars. It will be shown in section 3.6 that the log-likelihood of the model after training for the DDHMM is higher than the HMM log-likelihood by a factor of 3 implying the DDHMM is able to predict the road sequence with higher confidence.

Figures 3.11 and 3.12 shows frames of video from the Fields Point and Tehachapi sequences, two scenes consisting of conventional wind turbines. The periodic motion of the wind turbines creates oscillating intensity sequences at the highlighted pixels as seen in figures 3.11d and 3.12d. Again, in both scenes, neither the GMM or HMM can produce sample intensity sequences that mirror the dynamics of the original intensity sequences. However, intensity sequences sampled from the proposed DDHMM reflect the transitions between long sequences of sky to a short duration blade
back to long sky intensity sequences exhibited in the original data.

The Helix video sequences shown in figure 3.13 has a helix wind turbine in the foreground which produces a more complex but still periodic intensity waveform as seen in figure 3.13d. Again the temporal independence assumptions of the GMM and HMM models cause each to fail to capture relevant intensity duration information that the DDHMM both learns and mimics in the intensity sequence samples.

Note the proposed algorithm learns state duration and transition distributions using a single pass of the training sequence, while the EM algorithm uses an iterative update scheme and multiple passes to find locally optimal parameter settings in the case of the GMM and HMM models. However, without exception, the sampled intensity sequences produced by the DDHMM in each case more closely resemble the original waveform. These experiments reinforce the idea that no matter how well a particular algorithm can learn the parameters of a given model, if that model lacks the necessary context, such as not modeling state transition likelihood in the case of the GMM or state duration probabilities as in the case of the HMM, the model will fail to predict significant patterns in the data.
Figure 3.9: Swing video posterior sampling evaluation. The intensity sequence observed at the pixel location highlighted in figures 3.9a, 3.9b, and 3.9c is shown in figure 3.9d. Figures 3.9e and 3.9f show intensity sequence samples drawn from GMMs and HMMs respectively, trained using the Expectation Maximization algorithm for various numbers of states $K$. Figure 3.9g shows sample intensity sequences drawn from a DDHMM trained using the learning algorithm proposed in section 3.3.
Observed intensity sequence

Intensity sequences sampled from GMMs with $K \in \{1, \ldots, 8\}$

Intensity sequences sampled from HMMs with $K \in \{2, \ldots, 9\}$

Intensity sequences sampled from a DDHMM trained according to section 3.3

Figure 3.10: Highway posterior sampling evaluation. The intensity sequence extracted from the Highway video sequence at the pixel location at the center of the circle highlighted in figures 3.10a, 3.10b, and 3.10c is shown in figure 3.10d. The observed intensity sequence was used to train GMMs and HMMs for various number of states, $K$, using the Expectation Maximization algorithm. Figures 3.10e and 3.10f show intensity sequences sampled from the learned GMM and HMMs respectively. These samples are compared to the intensity sequences sampled from the DDHMM model and learning algorithm proposed in section 3.3 which are shown in figure 3.10g.
Figure 3.11: Fields Point posterior sampling evaluation. The intensity sequence extracted from the Fields Point video sequence at the pixel location at the center of the circle highlight in figures 3.11a, 3.11b, and 3.11c is shown in figure 3.11d. The extracted intensity sequence was used to train GMMs and HMMs for various number of states, $K$, using the Expectation Maximization algorithm. Figures 3.11e and 3.11f show intensity sequences sampled from the learned GMM and HMMs respectively. These samples are compared to the intensity sequences sampled from the DDHMM model and learning algorithm proposed in section 3.3 and shown in figure 3.11g.
Figure 3.12: Tehachapi posterior sampling evaluation. The intensity sequence observed in the Tehachapi video sequence at the pixel location at the center of the circle in figures 3.12a, 3.12b and 3.12c is shown in figure 3.12d. Intensity sequences sampled from GMMs and HMMs with various number of states, trained using the observed sequence and the Expectation Maximization algorithm are shown in figures 3.12e and 3.12f. Samples of intensity sequences generated by a DDHMM trained according to the algorithms in section 3.3 are shown in figure 3.12g for comparison.
Figure 3.13: Helix posterior sampling evaluation. The intensity sequence observed in the Helix video sequence at the pixel location at the center of the circle in figures 3.13a, 3.13b, and 3.13c is shown in figure 3.13d. Intensity sequences sampled from GMMs and HMMs with various numbers of states, $K \in \{1, \ldots, 8\}$, trained using the Expectation Maximization algorithm are shown in figures 3.13e and 3.13f. Samples of intensity sequences generated by a DDHMM trained according to the algorithms developed in section 3.3 are shown in figure 3.13g for comparison.
3.5 Empirical DDHMM Complexity

This section examines the complexity of the per-pixel DDHMM models when trained on real-world surveillance videos. Statistics of per-pixel DDHMM model complexity after training models using the proposed learning algorithm on real-world surveillance video are compiled. Additionally, correlations between per-pixel entropy and model complexity are examined.

The Shannon-entropy, or entropy for short, of a discrete probability distribution $p(X)$ is defined mathematically as [68]

$$- \sum_{x} p(x) \log_2 p(x)$$

(3.12)

The entropy of a distribution may be interpreted as a measure of predictability of the random variable, with larger values of entropy implying less predictability. To measure the entropy at each pixel location in a frame of video, a histogram of intensity values was computed using the values observed at every time step for each pixel location. Using the histogram to approximate the intensity distribution, entropy was computed at each pixel according to equation 3.12. After training per-pixel DDHMMs on the corresponding video sequence, the integer number of states created at each pixel location was recorded. The number of states created by the DDHMM is more closely correlated with the number of distinct modes of the static intensity distribution associated with a pixel location, however entropy is extremely easy to compute compared to mode-estimation techniques and does offer insights about the pixel locations that require more DDHMM states to model accurately.

Figure 3.14 visualizes the per-pixel entropy and the number of states created for each per-pixel DDHMM trained using the Fields Point video. In figure 3.14a, there are large values of entropy in areas of the frame corresponding to the foliage and chain link fence which move randomly due to the wind. Subsequently, there are 2-3 states created at these locations as seen in figure 3.14b. Similarly, pixel locations which correspond to the turbine blades exhibit large values of entropy, decreasing with distance from the turbine center. The true center of the turbine retains a small near-constant intensity value. However, just off center the angular speed of the blade is greatest, and there is a
nearly uniform oscillation between the sky and blade intensity corresponding to low predictability and high entropy. As the radial distance with respect to each turbine center increases, the orbital radial frequency of each blade decreases, increasing the number of observations of the sky intensity relative to the blade intensity. This results in an increased predictability of intensity values, the sky intensity is more common, thus the entropy decreases with radial distance. However, as seen in 3.14b, the number of states however remains relatively constant over the extent of pixel associated with the turbine. This is due to the fact that the number of states is better predicted by the number of dominant modes of the intensity distribution, regardless of the relative weight of the modes which along with the spread of the mode determines entropy.

(a) Entropy of empirical intensity distribution (b) Number of states at each per-pixel DDHMM at each pixel.

Figure 3.14: Entropy of empirical intensity distribution and number of states at each per-pixel DDHMM for the Fields Point video sequence

Complexity patterns similar to the Fields Point video are seen in the Helix and Tehachapi videos as shown in figures 3.15 and 3.16. The helical wind turbine produces intensity patterns with more modes and more states are created in these areas of the frame relative to the simpler conventional turbine in the Tehachapi video reflected in figures 3.15b and 3.16b.

Figures 3.18a and 3.18b show the per-pixel entropy and number of states created at each DDHMM in the Swing video. The mother holds the child for a few seconds prior to letting the child swing in each swing cycle. This means the pixels corresponding to the swing set closest to the mother exhibit an intensity distribution which is more uniformly distributed between the child and background
(a) Entropy of empirical intensity distribution (b) Number of states at each per-pixel DDHMM at each pixel.

Figure 3.15: Entropy of empirical intensity distribution and number of states at each per-pixel DDHMM for the Helix video sequence.

Intensities. Other pixel locations still in the path of the swinging child but further from the mother observe intensities associated with the child infrequently compared to the background, resulting in lower entropy. Additionally, the mother remains in a relatively fixed position with slight inter-frame intensity perturbations along her outline. This results in areas of high entropy at the edges of the mother but extremely low in the core areas which undergo little movement and exhibit relatively constant intensities over time. Again, the DDHMM creates 2-4 states to account for the intensity variations of the periodically reoccurring intensities associated with the child in the swing area, but only use 1-2 states to describe the nearly constant intensities associated with the pixels located on
the core of the mother. The relatively large number of states (these still are only roughly four or five states) directly behind the mother correspond to pedestrians the enter and exit the scene being the mother during training.

Figure 3.18: Entropy of empirical intensity distribution at each pixel and number of states at each per-pixel DDHMM for the Swing video sequence.

Figure 3.19 shows the entropy-complexity analysis for the Highway sequence. The pixel locations associated with the road exhibit high appearance variability and low predictability. The DDHMM learning algorithm allocates a relatively high number of states, four to six, to pixel locations corresponding to the road. While this area contains DDHMMs with the largest number of states across
the experiments, the comparatively irregular intensity sequences observed on the road warrant a more complex model. For example, figure 3.17 plots the empirical distribution of intensity values taken within the highlighted region on the road aggregated over every frame of video. By inspection, the intensity distribution has three dominant modes and significant mass spread uniformly over the region 25-125 gray values. The three modes induce at least 3 states while the remaining mass in the 25-125 region will require a few more high variance Gaussian states to approximate the flat intensity distribution. The Tehachapi intensity distribution, computed over all frames at pixel locations within the marked region in figure 3.17a, shown in figure 3.17b is much simpler when compared to the Highway example. There are two dominant modes which can be accurately accounted for by two Gaussian distributions which, according to figure 3.16b are correctly identified by the DDHMM learning algorithm.

![Entropy of empirical intensity distribution and number of states at each DDHMM for the Highway video sequence.](image)

(a) Entropy of empirical intensity distribution  
(b) Number of states at each per-pixel DDHMM at each pixel.

Table 3.3 shows the average number of states created at each pixel location in each video sequence. While real-world surveillance videos exhibit areas with high dynamic content that warrant complex models, most pixel locations are associated with static content and constant appearances. Therefore, the average DDHMM complexity as measured by the number of states created by each model is relatively low, around 2-3 states per scene.
(a) Spatial area used to compute intensity distribution.

Figure 3.20: Empirical intensity distribution computed over all frames at pixel locations within the highlighted section.

(b) Empirical intensity distribution taken over all frames within highlighted road segment.

<table>
<thead>
<tr>
<th>Video Sequence</th>
<th>Average number of states/pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swing</td>
<td>2.0</td>
</tr>
<tr>
<td>Fields Point</td>
<td>2.0</td>
</tr>
<tr>
<td>Helix</td>
<td>2.0</td>
</tr>
<tr>
<td>Tehachapi</td>
<td>1.0</td>
</tr>
<tr>
<td>Highway</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Table 3.3: Average number of DDHMM states created, rounded to the nearest integer, for each video sequence.

### 3.6 DDHMM Evaluation: Likelihood

This section evaluates the predictive power of the proposed DDHMM and learning algorithm by comparing training a GMM, HMM and DDHMM with common training intensity sequences and evaluating the log-likelihood of each model relative to a test intensity sequence from the same scene.

The log-likelihood of the GMM, HMM and DDHMMs are given by

\[
\ln \mathcal{L}(y_1, \ldots, y_T) = \sum_{t=1}^{T} \ln \left[ \sum_{k=1}^{K} p(s_t) p(y_t | s_t) \right]
\]

\[
\ln \mathcal{L}(y_1, \ldots, y_T) = \ln \left[ \sum_{s_1, \ldots, s_T} p(y_1, \ldots, y_T, s_1, \ldots, s_T) \right]
\]

\[
\ln \mathcal{L}(y_1, \ldots, y_T) = \ln \left[ \sum_{s, D} p(y_1, \ldots, y_T, S_1, \ldots, S_L, D_1, \ldots, D_L) \right]
\]
where the joint densities of observations and latent variables in equations 3.14 and 3.15 were given in equations 2.17 and 3.4. The summations in equations 3.14 and 3.15 are taken with respect to all possible latent state-duration configurations which implicitly marginalizes over all possible number of segments, $L$. In equation 3.15, the summation over duration random variables assumes the range of the duration variable is restricted to a finite interval.

The GMM log-likelihood can be computed directly in $\mathcal{O}(TK)$ time as the logarithm of the marginal observation density turns a product into a sum that factors over all observations. As previously mentioned, the log-likelihood according to the HMM is more difficult to compute. Direct marginalization of the joint distribution over latent states would scale as $\mathcal{O}(K^T)$ which would be impossible except for trivially short observation sequences. However, a dynamic programming algorithm known as the backward-forward algorithm [56, 45] achieves exact results in $\mathcal{O}(TK^2)$ time, linear in the length of the observation sequence, by saving and reusing partial summations. The forward-backward algorithm is commonly used as a subroutine within the Expectation Maximization for estimating the parameters of a HMM as in section 3.4. A review of the forward-backward algorithm is given in appendix A and the listed forward-backward subroutines were used to evaluate the HMM log-likelihood of test sequences in this section.

### 3.6.1 Forward program to compute DDHMM log-likelihood

A forward-backward recursion for computing the posterior latent state probabilities and model likelihood has been developed for several types of HSMMs [56, 69, 70, 71, 72, 65]. This section develops a dynamic program for computing the log-likelihood of an observation sequence given a particular setting of model parameters for the DDHMM. Alternative forward-backward algorithm derivations exist in the DDHMM literature, however because the goal is to only compute the log-likelihood of the DDHMM for a fixed parameter setting, a simpler more direct derivation of the forward program is presented here.

Define $T$ forward variables such that at a time $t$, the forward variable $\alpha_{tk}$ is the joint probability
of the observations up to time $t$ and a transition occurs from any state at time $t$ to state $s_k$ at time $t+1$.

$$\alpha_{tk} = p(q_{t+1} = s_k, q_t \neq s_k, y_1, \ldots, y_t) \quad (3.16)$$

where the base case is defined as

$$\alpha_{1k} = \sum_{\forall j \neq k} p(s_k | s_j) p(y_1 | \theta_j) p(q_t = s_j) \quad (3.17)$$

Each forward variable may be computed recursively in terms of the previous forward variables

$$\alpha_{tk} = \sum_{\forall j \neq k} \sum_{t'-d_{\text{min}}}^{t-d_{\text{max}}} \alpha_{t'j} p(D = t - t' | s_j) p(s_k | s_j) \prod_{z=t'}^{t} p(y_z | \theta_k) \quad (3.18)$$

where $d_{\text{min}}$ and $d_{\text{max}}$ are the shortest and longest allowable state durations.

The last forward variable takes into account the fact that observations may have been cut-off during a state segment and the duration of the last state may have lasted beyond the recorded observations. Therefore, the forward variable at the last time step takes into account the state survival distribution to express the uncertainty associated with the duration of the final state

$$\alpha_{Tk} = p(q_{T+1} = s_k, q_T \neq s_k, y_1, \ldots, y_T)$$

$$\alpha_{Tk} = \sum_{t=1}^{T} \alpha_{tk} p(D \geq T - t + 1) p(y_t, \ldots, y_{T-t+1} | \theta_k) \quad (3.19)$$

For example, consider computing the last forward variable for a sequence of observations with length $T = 3$. Expanding equation 3.19

$$\alpha_{Tk} = \alpha_{0k} p(D \geq 3 | q = s_k) \prod_{t=1}^{T} p(y_t | \theta_k)$$

$$+ \alpha_{1k} p(D \geq 2 | q = s_k) \prod_{t=2}^{T} p(y_t | \theta_k)$$

$$+ \alpha_{2k} p(D \geq 1 | q = s_k) p(y_T | \theta_k) \quad (3.20)$$

The forward variables $\alpha_{0k}$, $\alpha_{1k}$, and $\alpha_{2k}$ by definition are the joint probabilities of previous observations and that there are transitions to state $k$ at times 1, 2 and 3 respectively. The forward variable $\alpha_{Tk}$ is then the sum of probabilities there is a transition to state $s_k$ at any time and state $s_k$ persists
past the length of the chain. The likelihood of the DDHMM would then be the sum of the final forward variable for each possible state, marginalizing over all possible state transitions

$$L(y_1, \ldots, y_T) = \sum_{k=1}^{K} \alpha_{Tk}$$ (3.21)

To compute the forward variable at each step requires $O(K^2 \min(t, |D|)) \rightarrow O(K^2|D|)$, implying running the forward program to conclusion will take $O(TK^2|D|)$ time. In all experiments $|D| = T$, therefore to compute the $K$ forward variables requires $O(T^2K^2)$ operations.

### 3.6.2 Results

For each video sequence listed in table 3.4, a GMM, HMM, and DDHMM were trained using the same intensity sequence extracted from a pixel location in each video using the EM algorithm and the proposed methods. A second intensity sequence from the same pixel location was used to evaluate the log-likelihood of the sequence according to each model which are shown in table 3.4. The GMM log-likelihood was computed directly, the HMM likelihood was computed using the standard forward program outlined in appendix A and the DDHMM log-likelihood was computed according to the methods developed in section 3.6.1. A model log-likelihood score closer to zero implies the data is better explained by a given model. Also note that the results in table 3.4 are on a log scale and imply an increase in the likelihood by a factor of 2-3 between the HMM and DDHMM and 4-5 between the GMM and DDHMM.

<table>
<thead>
<tr>
<th>Log-Likelihood</th>
<th>GMM</th>
<th>HMM</th>
<th>DDHMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highway</td>
<td>-857</td>
<td>-716</td>
<td><strong>-703</strong></td>
</tr>
<tr>
<td>Helix</td>
<td>-814</td>
<td>-685</td>
<td><strong>-670</strong></td>
</tr>
<tr>
<td>Tehachapi</td>
<td>-246</td>
<td>-162</td>
<td><strong>-146</strong></td>
</tr>
</tbody>
</table>

Table 3.4: Log-likelihood of the test observation sequences according to each model. Values closer to zero are desirable. Note that these measurements are on the log scale and therefore represent an increase in likelihood by a factor of 2-3 when comparing the HMM and DDHMM.
3.7 Duration Dependent Codebook Classifier

While the probability of an intensity sequence at any pixel could be computed using the DDHMM framework, the probability decays exponentially with the length of the intensity sequence. Thus, some form of length normalization procedure is needed to define a consistent threshold, and it is unclear what principle can be used to define such a normalization under all situations. Therefore, a code book based approach is taken to compare atomic elements of intensity sequences obtained from the implicit temporal segmentation provided by the DDHMM state assignments described in section 3.3.

The DDHMM learning algorithm described in Section 3.3 is run continuously during the training and testing phases to associate every pixel intensity observation with a DDHMM state label. The resulting per-pixel state assignments are used to compose code words of the form \((s_p, d, s_n)\); the previous state label, the duration of \(s_p\) and the next state, respectively. The user provides the system with video footage containing normal scene dynamics and the code words produced at each pixel are recorded in a codebook (one for every pixel). Any code word observed during testing that does not exist in the code book is considered a deviation from normal scene dynamics and is flagged as a change of interest. In all experiments, the duration \(d\) was further quantized into the ranges \((1, 6), [6, 10), [10, 25), [25, \infty))\) to further compress the per-pixel codebooks and a 7 \(\times\) 7 median filter was applied to each frame of the change detection output as a post-processing step.

3.8 Change Localization by State Merging and Splicing

Under the proposed model, normal background sequences can exhibit long DDHMM state duration times: \((s_t, d_t)\), which may persist for hundreds of frames. If a change occurs in the middle of such normal state durations then the entire period is considered to be change. This error occurs because no normal background states possesses the appropriate intensity distributions with shorter durations on each side of the change time interval. These false positive errors can be eliminated by "splicing".
The localization module does not modify the underlying state assignment but merely suppresses false positive artifacts arising from changes in the scene disrupting the duration of otherwise normal states and helps localize the change point in time. Where the change interval is replaced by the normal background state, with the duration that spans the full interval. Thus, the intensity observations outside the change interval are considered normal even given the splitting of the normal state duration by changes. Figure 3.21 illustrates the splicing procedure, where there are three cases to consider: left merge - the change interval starts at the same time as the long duration normal state; right merge - the change interval ends at the same time as the normal state duration; middle-splice - the change is fully interior to the long duration state time interval.

3.9 Change Detection Results

Figures 3.22 and 3.23 show example frames from the Fields Point Turbine and Swing video sequences respectively. The images in the second row visualize hand labeled ground truth corresponding to the example video frame in the same column. A white pixel represents a true positive, black true negative, and gray is unknown due to shadow or ambiguous object boundaries. In both figures the last three columns show the detections made by the proposed method and the reviewed state-of-the-art algorithms. Again, a white pixel means the algorithm labeled the observation as a change of interest and a black pixel is the contrary.

The Fields Point scene in Figure 3.22 monitors the entrance to a facility with wind turbines
spinning in the background. Cars passing on the road are considered a change of interest whereas the spinning blades are a normal dynamic feature in the scene. The proposed method is able to model the spinning turbine, avoiding false positives resulting from normal blade movement which the other state-of-the-art algorithms mistake as meaningful change.

The Swing video sequence shown in Figure 3.23 shows a mother pushing her daughter on a swing set and eventually, a previously unobserved pedestrian enters and exits the scene. This seemingly innocuous footage contains interesting periodic phenomena that modern change detection algorithms cannot model. The mother’s motions are repetitive as she pushes the child with a periodic rhythm. The mother and daughter on the swing set are considered normal, they are using the swing set for the entirety of the video sequence, and the pedestrian is a change of interest.

The change detection results for the proposed method show that the system is capable of learning the normal intensity sequences associated with the swinging child and mother by correctly labeling their pixel locations in the scene as normal. The system also correctly labels the majority of the pedestrian as change. The other state-of-the-art methods have no mechanism to associate observations caused by periodic motion with normal scene activity. To note one limitation of the proposed system, there are some false negatives on the pedestrian’s legs. The pedestrian’s pants are a similar color to the black harness of the swing that moves with comparable velocity, resulting in coincident state sojourn times. These mistakes are to be expected using only per-pixel intensity observations.

It may be argued that the parameters of the competing algorithms could be adjusted to better suit this scene. For example, it is possible to tune PBAS to maintain more background samples, allowing it to keep track of reoccurring pixel intensities that occur over a larger time scale. However, this global parameter adjustment greatly increases the storage and computational cost of PBAS. In contrast, the proposed method automatically adjusts the complexity of each DDHMM locally to the demands of the observations at each pixel. The algorithm adapts the complexity of each per-pixel intensity representation while the underlying DDHMM state-duration assignments which are used as codewords, compactly encodes the appropriate time scale for repeating intensity sequences.
(a) Example Frames
(b) Ground Truth
(c) Proposed
(d) SG-GMM
(e) CBBGS
Figure 3.22: Fields Point: Frames of video of the Fields Point sequence are shown in the first row with time flowing from left to right across the columns. The second row shows hand-labeled ground truth masks corresponding to the frame in the same column. The third through ninth rows show the change detection results of the proposed, SG-GMM, CBBGS, STLBP, PBAS, ViBe, STLBP and SGMM-SOD algorithms for the example frame in the corresponding row. The proposed method is the only algorithm able to learn that the spinning blades are a normal part of the scene and can still detect the previously unobserved cars. The results of the SG-GMM algorithm were obtained using the implementation provided by [20].
(a) Example Frames

(b) Ground Truth

(c) Proposed

(d) SG-GMM

(e) CBBGS
Figure 3.23: Swing sequence: Change detection results for algorithms run on the Swing video are shown as binary masks. A white pixel indicates change was detected at the location whereas black is no change. Each row shows the output of a different change detection algorithm for the example frames in figure 3.23a where time flows from left to right. Hand-segmented groundtruth is shown in figure 3.23b and the output of the proposed DDHMM codebook, SG-GMM, CBBGS, PBAS, ViBe, STLBP, SGMM-SOD algorithms are shown in the remaining rows. The proposed method is the only algorithm that learns the swinging child is a normal part of the scene but still detects the previously unobserved pedestrian. The results of the SG-GMM algorithm were obtained using the implementation provided by [20].

Tables 3.5 and 3.6 quantify the performance of the proposed and competing algorithms for the Fields Point and Swing video sequences respectively. The competing algorithms contain a large
number of false positives in every frame of video for both scenes. The blades of the turbine in the Fields Point video and the swinging child in the Swing scene are always incorrectly labeled as change and thus report each frame of video to the end user for further inspection. These algorithms would require additional reasoning modules to reliably filter their output for use in a surveillance application. However, the proposed method is able to discriminate between the normal repeating phenomena and previously unobserved events resulting in superior precision and false positive rate.

<table>
<thead>
<tr>
<th>Method</th>
<th>TPR</th>
<th>Precision</th>
<th>FPR</th>
<th>FNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBAS</td>
<td>99.3%</td>
<td>68.8%</td>
<td>0.82%</td>
<td>0.012%</td>
</tr>
<tr>
<td>ViBe</td>
<td>90.5%</td>
<td>54.7%</td>
<td>1.37%</td>
<td>0.174%</td>
</tr>
<tr>
<td>STLBP</td>
<td>91.9%</td>
<td>38.5%</td>
<td>2.68%</td>
<td>0.148%</td>
</tr>
<tr>
<td>SGMM-SOD</td>
<td>96.5%</td>
<td>67.2%</td>
<td>0.86%</td>
<td>0.064%</td>
</tr>
<tr>
<td>CBBGS</td>
<td>91.8%</td>
<td>40.2%</td>
<td>2.49%</td>
<td>0.149%</td>
</tr>
<tr>
<td>Proposed</td>
<td>98.6%</td>
<td>82.2%</td>
<td>0.39%</td>
<td>0.025%</td>
</tr>
</tbody>
</table>

Table 3.5: *Fields Point:* Quantitative change detection results showing True Positive Rate (TPR), Precision, False Positive Rate (FPR) and False Negative Rate (FNR).

<table>
<thead>
<tr>
<th>Method</th>
<th>TPR</th>
<th>Precision</th>
<th>FPR</th>
<th>FNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBAS</td>
<td>89.3%</td>
<td>30.2%</td>
<td>2.42%</td>
<td>0.13%</td>
</tr>
<tr>
<td>ViBe</td>
<td>80.8%</td>
<td>17.0%</td>
<td>4.63%</td>
<td>0.23%</td>
</tr>
<tr>
<td>SPLBP</td>
<td>76.6%</td>
<td>8.6%</td>
<td>9.55%</td>
<td>0.27%</td>
</tr>
<tr>
<td>SGMM-SOD</td>
<td>92.2%</td>
<td>23.9%</td>
<td>3.45%</td>
<td>0.09%</td>
</tr>
<tr>
<td>CBBGS</td>
<td>84.9%</td>
<td>30.8%</td>
<td>2.25%</td>
<td>0.18%</td>
</tr>
<tr>
<td>Proposed</td>
<td>80.7%</td>
<td>41.1%</td>
<td>1.36%</td>
<td>0.23%</td>
</tr>
</tbody>
</table>

Table 3.6: *Swing:* Quantitative change detection results showing True Positive Rate (TPR), Precision, False Positive Rate (FPR) and False Negative Rate (FNR).
Chapter 4

Region based change detection via change point geometry

The DDHMM based change detection method proposed in chapter 3 assumes intensity sequences at each pixel location are independent. As such, the algorithm is easy to parallelize and implement for real-time change detection. However, pixel intensity sequences are rarely independent. Object motion captured across a series of video frames creates correlated intensity sequences and real-world objects typically exhibit color and intensity patterns that are spatially related. The proposed pixel independent duration dependent hidden Markov models more accurately model temporal patterns than previous state-of-the-art per-pixel models but sacrifice spatial information for simplicity. This chapter investigates how to incorporate region-level reasoning, spatial and temporal information, with per-pixel duration dependent hidden Markov models to extract higher level semantic information from surveillance video than was possible under the pixel-independent assumptions of chapter 3.

An algorithm is proposed based on geometric descriptors of space-time appearance discontinuities for fixed camera surveillance video. At each pixel in a video frame, intensity subsequences with similar appearance and duration statistics are segmented using the duration dependent Markov
model developed in chapter 3. The start of each per-pixel homogeneous subsequence are referred to as change point vertices. A change point vertex is a 3-d point in 2-d image space-time and represents the start of an approximately homogeneous intensity sequence that differs significantly from the previous segment. Change point vertices are then clustered using an efficient graph based segmentation algorithm to construct a change point hull. A change point hull may be thought of as a space-time "super-voxel" of discontinuities, demarcating regions of similar appearance and duration statistics that differ from their spatio-temporal neighbors. The initial segmentation retains the efficiency of the pixel level algorithm while the clustering step builds a rich, data driven space-time descriptor with minimal computational overhead. The geometry of the change point hull provides a discriminating feature for distinguishing coherent object movement from random or stochastic appearance dynamics and is simultaneously a rich descriptor for reasoning about object velocity and direction.

4.1 Related Region-Level Algorithms

4.1.1 Spatio-Temporal Local Binary Patterns (STLBP)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Summary</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{\text{lbp}}$</td>
<td>LBP neighborhood radius</td>
<td>9</td>
</tr>
<tr>
<td>$R_{\text{H}}$</td>
<td>LBP histogram neighborhood radius</td>
<td>6</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Weight parameter for temporal and spatial LBP code averaging</td>
<td>0.3</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of STLBP histograms in background model</td>
<td>6</td>
</tr>
<tr>
<td>$\alpha_{\text{init}}$</td>
<td>Initial STLBP histogram weight</td>
<td>0.01</td>
</tr>
<tr>
<td>$T_H$</td>
<td>Histogram intersection measure threshold</td>
<td>0.7</td>
</tr>
<tr>
<td>$\gamma_\alpha$</td>
<td>STLBP histogram counts learning rate</td>
<td>0.01</td>
</tr>
<tr>
<td>$\gamma_\omega$</td>
<td>STLBP histogram weight learning rate</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 4.1: STLBP algorithm parameter summary.

The method proposed in [22], refered to as STLBP in this thesis, uses a spatial-temporal region descriptor based on Local Binary Patterns (LBP) for detecting change in video. Local binary patterns, were originally introduced by [73] and [74] and were used to detect moving objects in [21].
The LBP descriptor encodes region statistics with a single number derived by thresholding the intensity values of pixels within a local spatial neighborhood by the intensity value at the center of the region. For a given frame of video, target location \( c = (x, y) \) and spatial neighborhood \( \mathcal{N}(c) = \{ n_i = (x_i, y_i) \mid i \in \{0, \ldots, N - 1\} \} \), a set of \( N \) pixel locations surrounding the target pixel, the LBP descriptor is computed as:

\[
LBP(c) = \sum_{i=0}^{N-1} \tau(I(n_i) - I(c)) 2^i
\]

(4.1)

Where \( I(n_i) \) intensity associated with the neighbor \( n_i \) and \( I(c) \) is the intensity associated with the target pixel. The function \( \tau(\cdot) \) is a binary threshold which takes the value of one if the argument is greater or equal zero and returns zero otherwise [22]. An example of computing the traditional LBP descriptor is shown in 4.1 which has been replicated from [21].

![Figure 4.1: Example computing the LBP descriptor at the center pixel using an eight pixel neighborhood where each pixel is labeled zero through seven. First, the intensity values at each neighbor is compared to the intensity value at the center pixel and each neighbor is set to the value of one if its intensity is greater than the intensity at the center pixel or zero otherwise. The results are then multiplied by two raised to the power of the neighbor index. Finally, the results from the previous step are summed over all neighbors to assign a single LBP code to the target pixel. In this example the final LBP code for the center pixel (shown in gray) is 143. This example originally appeared in [21].](image)

The basic LBP descriptor is extended by [22] which proposes a spatio-temporal texture descriptor that is a weighted sum of LBP histograms computed with the current frame at time index \( t \) and the previous frame at time index \( t - 1 \) of a video sequence. For a given target location in a frame
of video with time index $t$, let $c_t = (x_t, y_t)$ be a target pixel location in frame $t$. A spatial and a
temporal neighborhood are defined for each pixel location. Each neighborhood is an ordered list of
pixel locations within the radius $R_{\text{lpb}}$ of the target location $c_t$. The spatial neighborhood is a list
of locations in the same frame as $c_t$ defined as:

$$N_t(c_t) = \{ n_{(t,i)} = (x_{(t,i)}, y_{(t,i)}) \mid \sqrt{(x_{(t,i)} - x_t)^2 + (y_{(t,i)} - y_t)^2} \leq R_{\text{lpb}}, c_t = (x_t, y_t) \}$$  \hspace{1cm} (4.2)$$

Likewise, the temporal neighborhood of $c_t$ is defined as an ordered list of pixel locations within
radius $R_{\text{lpb}}$ of $c_t$ in the previous frame of video:

$$N_{t-1}(c_t) = \{ n_{(t-1,i)} = (x_{(t-1,i)}, y_{(t-1,i)}) \mid \sqrt{(x_{(t-1,i)} - x_t)^2 + (y_{(t-1,i)} - y_t)^2} \leq R_{\text{lpb}}, c_t = (x_t, y_t) \}$$  \hspace{1cm} (4.3)$$

The spatial and temporal neighborhoods of a pixel $c_t$ are used to compute the spatial and temporal
LBP codes defined as

$$\text{LBP}_t(c_t) = \sum_{\forall n_{i} \in N_t(c_t)} \tau (I(n_i) - I(c_t)) 2^i$$  \hspace{1cm} (4.4)$$

$$\text{LBP}_{t-1}(c_t) = \sum_{\forall n_{i} \in N_{t-1}(c_t)} \tau (I(n_i) - I(c_t)) 2^i$$  \hspace{1cm} (4.5)$$

Two histograms, a spatial and temporal histogram of LBP codes within the a radius $R_H$ of a target
pixel $c_t$ are created using the spatial and temporal LBP codes of pixels surrounding the target

$$H_{(t,j)} = \sum_{n_i \in N_{H}(c_t)} \delta (\text{LBP}_t(n_i) = j), j = 0, \ldots, 2^{N-1} \hspace{1cm} (4.6)$$

$$H_{(t-1,j)} = \sum_{n_i \in N_{H}(c_t)} \delta (\text{LBP}_{t-1}(n_i) = j), j = 0, \ldots, 2^{N-1} \hspace{1cm} (4.7)$$

$$N_H(c_t) = \{ n_{i} = (x_{(t,i)}, y_{(t,i)}) \mid \sqrt{(x_{(t,i)} + x_t)^2 + (y_{(t,i)} - y_t)^2} \leq R_H, c_t = (x_t, y_t) \}$$  \hspace{1cm} (4.8)$$

where $\delta (\cdot)$ is the indicator function, taking a value of one if the argument is true and zero otherwise.

The final STLBP descriptor is a histogram whose entries are the weighted average of the counts
from the spatial and temporal LBP histograms:

\[ \bar{H}_{(t,i)} = \omega H_{(t-1,i)} + (1 - \omega)H_{(t,i)} \]

\[ \omega \in [0, 1], \quad i \in \{1, \ldots, N - 1\} \]  \hfill (4.9)

To detect changes, the STLBP algorithm implements the background subtraction algorithm proposed by [21], substituting the augmented spatio-temporal LBP for the original LBP histograms. For each pixel location, a list of \( K \) STLBP histograms are maintained, each with a corresponding normalized weight. The array of STLBP histograms and their corresponding weights are referred to as the background model. Let \( \bar{H} \) be the array of \( K \) STLBP histograms and \( \alpha \) be the array of \( K \) weights such that

\[ \bar{H} = \{ \bar{H}_{(k,i)} \mid k \in \{1, \ldots, K\} \} \]  \hfill (4.10)

\[ \alpha = \left\{ \alpha_k \left| \sum_{k=1}^{K} \alpha_k = 1, \alpha_k \in [0, 1], k \in \{1, \ldots, K\} \right. \right\} \]  \hfill (4.11)

For each frame of video, the STLBP descriptor \( \bar{H}_{(t,i)} \) is compared to the \( K \) STLBP descriptors in the array using a histogram intersection measure

\[ \bar{H} \cap (\bar{H}_{(t,l)}, \bar{H}_{(j)}) = \sum_{i=0}^{N-1} \min \{ \bar{H}_{(t,i)}, \bar{H}_{(j,i)} \} \]  \hfill (4.12)

Similar to the philosophy of the SG-GMM algorithm, if there is no existing histogram in the background model array with an intersection measure within a threshold \( T_{\bar{H}} \), the histogram with the smallest weight is replaced by the current STLBP histogram with a small initial weight \( \alpha_{\text{init}} \). The resulting set of weights are subsequently re-normalized. If there are histograms with intersections distances less than the threshold, the histogram with the smallest intersection measure is updated according to:

\[ \bar{H}_{(k,i)} \leftarrow \gamma_\alpha \bar{H}_{(t,i)} + (1 - \gamma_\alpha) \bar{H}_{(k,i)} \]  \hfill (4.13)

\[ \alpha_k \leftarrow (1 - \gamma_\omega) \alpha_k + \gamma_\omega \]  \hfill (4.14)
If a STLBP histogram was found in the background model array which had an intersection distance less than the user-defined threshold $T_H$, the target pixel in the current frame is labeled as background. If no matching histogram was found in the background model, the target location is labeled as a significant change in the current frame.

Figure 4.2 shows the change detection results for the STLBP algorithm compared to the GMM and GKDE algorithms. The example frames in the first row show a scene with foliage in the background which change appearance over a short time scale as the leaves blow in the wind. Hand-labeled ground truth, shown in the second row, mark a pedestrian that enters the scene as change that are visualized as white pixels. The background, which is not considered meaningful change despite the movement of the foliage is visualized as black. The results of the STLBP, GMM and GKDE are visualized in the third, fourth and fifth rows respectively where change at a particular pixel is white and not-change is black.

While the STLBP algorithm avoids false positive detections from the dynamic background of the moving foliage that confuse the SG-GMM and GKDE algorithms, the local nature of the temporal neighborhood cannot model intensity observations that reoccur with a long period as shown in section 3.9. In figure 3.22h the STLBP algorithm was unable to model the rotating blade and as shown in figure 3.23h, failed to capture the periodic motion of the swinging child. The first row shows example frames from a video sequence. while the STLBP considers local region and temporal information, limited spatio-temporal extent of the descriptor fails to capture higher order patterns in video data.

The STLBP descriptor is a similar feature compared to the change point geometric descriptor proposed in this chapter but there are major differences. First, the STLBP algorithm defines a fixed local neighborhood for computing a gradient-like feature centered at each pixel whilst the proposed algorithm uses a data driven approach to directly model the natural scale of homogeneous sequences and their corresponding discontinuities. Secondly, the STLBP algorithm summarizes local texture using a histogram of LBP codes, taking a bag-of-words approach. The shape of the
appearance discontinuities is therefore lost and with it, the ability to reason about higher level motion or behaviors occurring in target videos. The proposed method however, uses the shape of appearance discontinuities as a fundamental feature for local scene description and change detection.
<table>
<thead>
<tr>
<th>Parameter/Notation</th>
<th>Summary</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_S$</td>
<td>Short-term per-pixel background model</td>
<td>N/A</td>
</tr>
<tr>
<td>$B_L$</td>
<td>Long-term per-pixel background model</td>
<td>N/A</td>
</tr>
<tr>
<td>$F_S$</td>
<td>Foreground detection mask associated with every per-pixel $B_S$</td>
<td>N/A</td>
</tr>
<tr>
<td>$\alpha_L$</td>
<td>Learning rate for long-term background GMM</td>
<td>0.001</td>
</tr>
<tr>
<td>$\alpha_S$</td>
<td>Learning rate for short-term background GMM</td>
<td>0.01</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of Gaussian components for short- and long-term GMMs</td>
<td>5</td>
</tr>
<tr>
<td>$T$</td>
<td>Background component threshold</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 4.2: SGMM-SOD algorithm notation and parameter summary.

4.1.2 Dual Model Gaussian Mixtures (SGMM-SOD)

The SGMM-SOD algorithm introduced in [75] and expanded by [9] classifies each pixel based on two GMM models based on the original SG-GMM algorithm to region-level analysis. At each pixel, the short-term background model, denoted $B_S$, is the original Stuaffer-Grimson Gaussian mixture model [1] with a short learning rate $\alpha_S$ and the long-term background model, referred to as $B_L$, is a SG-GMM model with identical parameters modulo the learning rate, $\alpha_L$ which is ten times slower. A summary of the notation and parameters of the SGMM-SOD algorithm is provided by table 4.2.

While the SGMM-SOD algorithm introduces multiple modules for detecting change including, a lighting model and additional splitting/merging heuristics that augment the original SG-GMM Gaussian mixture model learning equations, the most fundamental innovation proposed by [9] is coarse pixel-level classification of change types using two GMM models with different learning rates and finite state machines combined with region level and matching analysis for detecting meaningful change. The proceeding review focuses on the dual model GMM, finite state machine reasoning for change type classification, and region matching modules while the details of the remaining contributions are left to the original source [9].

Let $F_S$ and $F_L$ be binary masks for each frame for the change detection results of the short- and long-term background models with a one indicating a significant change at a given pixel location and zero otherwise. The foreground masks for the short- and long-term models are used to classify observations into four categories: Background (BG), Moving Foreground MO, New-Static Foreground
(NS), and Uncovered Background (UBG) using a finite state machine. The finite state machine is defined as a five-tuple: \((I, Q, Z, f, O)\). \(I\) is the finite state machine alphabet corresponding to the four possibilities of the state of the short- and long-term foreground masks at each pixel: 
\[
I = \{ \{F_L, F_S\} \in \{(0,0), (0,1), (1,0), (1,1)\} \}.
\]

\(Q\) is an association between the finite state machine alphabet \(I\) and the semantic meaning of each detection set: \(Q \in \{BG, MO, ST, UBG\}\). \(Z\) enumerates the states of the finite state machine, \(Q\), and the relationships between \(I\), \(Q\) and \(Z\) are summarized in table 4.3. \(f\) and \(O\) are the next-state and output functions of the finite state machine and each take a value in \(Z\).

<table>
<thead>
<tr>
<th>(I) : ({F_L, F_S})</th>
<th>(Q)</th>
<th>(Z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>({0,0})</td>
<td>BG</td>
<td>0</td>
</tr>
<tr>
<td>({0,1})</td>
<td>UBG</td>
<td>1</td>
</tr>
<tr>
<td>({1,0})</td>
<td>ST</td>
<td>2</td>
</tr>
<tr>
<td>({1,1})</td>
<td>MO</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.3: Key variables for the per-pixel finite state machine classifier. \(I\) represents the short- and long-term per-pixel classification, \(\{F_L, F_S\}\), at a given pixel. \(Q\) are the semantic interpretations of the dual model detections: Background, Moving Object, New Static Foreground, and Uncovered Background and \(Z\) enumerates the states of \(Q\) [9].

The type of change or not-change observed at each pixel are classified by the finite state machine by reasoning about the four possible configurations of the short- and long-term foreground masks. If both models do not detect any change in a given frame at a particular location, \(I = \{0,0\}\), the observation is associated with the background state \(Q = BG\). It is assumed that moving objects will trigger positive detections in both the long- and short-term models: \(I = \{1,1\}, Q = MO\).

A positive short-term and negative long-term detection, \(I = \{0,1\}\), is assumed to be caused by uncovering the original background that had been occluded by a foreground object \(Q = UBG\). It is argued that there are two phenomena that could be responsible for positive detection by the long-term model but negative the short-term, \(I = \{1,0\}\). Either a new object was left behind in the scene long enough to be accounted for by the short-term but not the long-term models, or an object has been removed from the scene and the short-term model has absorbed the new background.
appearance which the long-term hasn’t yet $Q = ST$.

Static foreground regions, pixels classified as $Q = ST$, are grouped together and a bounding box is computed with respect to the detections. A static region is considered to be well-formed if it is stationary in position and size over time as computed by the bounding box and if it is isolated, meaning the pixels along the border of the bounding box are classified as background $Q = BG$. The edges of the static foreground region are compared to the long-term background model $B_L$ using Chamfer matching [76]. If the static region can be matched to the long-term background model, the change is classified as a removed object. If a detected region can be matched to the input frame, the region is classified as a new static object.

Figure 4.3 shows an example of the SGMM-SOD algorithm classifying different types of change regions. A pedestrian enters the scene with a rolling bag of luggage which is positively detected as change by the short and long term models. When the pedestrian leaves the luggage behind, the bag is absorbed by the short-term model but not by the long-term model. The bounding box of the object is stationary and surrounded by pixels associated with the background, leading to a well-formed region of change. It is reasoned that the change belongs to a new static object as visualized in yellow. When the luggage is eventually removed, the edges of the change region matches the long-term model and the region is classified as a removed object which is shown in gray.

While the SGMM-SOD algorithm accurately classifies and localizes changes in a wide range of scenarios, the algorithm fails to account for normal periodic appearance dynamics as was shown in the experiments in section 3.9. The SGMM-SOD algorithm would need to be augmented with high-level recognition modules to correctly account for periodic phenomena. For example, when the turbine blade moves between frames in the Fields Point sequence, the set of pixels associated with the turbine blade in the previous frame are classified by SGMM-SOD as a static object that has been removed and the set of pixels in the current frame which have just changed appearance to reflect the blade detect are classified as a new static foreground object. To correctly identify the sequence of static object removal and new foreground object insertion as belonging to a class of regular periodic
Figure 4.3: SGMM-SOD region classification example. Walking pedestrians are detected by both the long and short term background models and are displayed in green. The luggage initially left behind by the pedestrian in the middle frame is absorbed into the short term model but detected as change by the long term model. The bounding box is well-formed and therefore classified as a new static object visualized in yellow. When the luggage is removed, the edges of the change region match the long-term background model and the change is classified as a removed object shown in gray. Images are from [9].

phenomena would require the addition of an extra reasoning module. This module would need to remember examples of objects that have entered and exited scene locations and perform an extra matching step to recognize the underlying object or sequence of objects. However, the methods proposed in chapter 3 and in the current chapter, directly model periodic phenomena at the pixel level, avoiding the need for extra reasoning to account for complex appearance dynamics.

4.2 Change Point Geometry

The proposed region-level algorithm, originally published in [77], is visualized in figure 4.4. First, the DDHMM algorithm described in chapter 3 is used to segment per-pixel observations into temporal regions with similar appearance and duration. At each pixel, the start of a segmented intensity subsequence, a 3D point in 2D space-time known as a change-point vertex, marks the beginning
Figure 4.4: Visualization of proposed algorithm. Each pixel of an input video is segmented into intensity subsequences with similar appearance and duration using a Hidden Semi-Markov Model. The beginning of each subsequence, a 3D point in 2D space-time and known as a change point vertex, are clustered into groups with similar appearance using a graph segmentation algorithm to form a change point hull. The vertices within a change point hull are used to construct a spatial covariance matrix which is decomposed via Singular Value Decomposition (SVD) to obtain the eigenvector and eigenvalues of the change point hull. The eigenvectors and eigenvalues are then used as descriptors for coherent motion and detecting meaningful scene aberration.

An efficient graph-based clustering algorithm [23] is used to group neighboring change-point vertices based on the expected appearance of the temporal segment to form a change-point hull. A change point hull, $\Phi$ is therefore a finite collection of change point vertices:

$$\Phi = \{v_z \mid z \in \{1, 2, \ldots, \}\}$$  \hspace{1cm} (4.16)

Having established the concepts of change point vertices and change point hulls, change point geometry may be defined as the analysis of shapes implied by collections of change point vertices or hulls.

The clustering algorithm of [23] is formulated as a graph-segmentation problem that partitions nodes of the graph into different groups based on graph node similarity expressed as weighted edges or connections between nodes. Therefore, a similarity relation between two change vertices is

$$C = \begin{bmatrix}
\sum_{x} x^2 & \sum_{x} x y & \sum_{x} x t \\
\sum_{x} y x & \sum_{y} y^2 & \sum_{y} y t \\
\sum_{x} t x & \sum_{y} t y & \sum_{t} t^2
\end{bmatrix}$$

$$W = \begin{bmatrix}
w_0 & w_1 & w_2
\end{bmatrix}$$
defined as the weighted sum of the absolute difference between the means associated with the state conditional emission densities of each vertex and the absolute difference of the duration assignment corresponding to each vertex:

$$\Delta (v_1, v_2) = \sqrt{\gamma |\mu_1 - \mu_2| + (1 - \gamma) |d_1 - d_2|}$$ (4.17)

where $\mu_1$, $\mu_2$, $d_1$, and $d_2$ are the means associated with the Gaussian conditional emission densities of the states assigned to the first and second change point vertex and the duration values of the first and second state assignment respectively. $\gamma$ is a parameter weighting the influence of appearance and duration statistics. $\gamma$ is set to 0.5 in all experiments. A weighted undirected graph is constructed based on the location of the change point vertices. Each vertex is connected to its four nearest neighbors and the similarity measure of equation 4.17 is used to assign a weight to each edge.

After segmentation, the change-point hulls could be thought of as space-time "super-voxels" of discontinuities, collections of intensity sequences with similar appearance and duration statistics that differ significantly from neighboring collections of intensity sequences. The initial segmentation retains the efficiency of pixel level algorithms while the clustering step builds a rich, data-driven space-time descriptor with minimal computational overhead. The shape of the change-point hulls (clusters) provide a discriminating vocabulary for describing coherent motion in video sequences.

As a motivating example, consider the "Crossing Paths" sequence of fig. 4.5. In this video, a camera has been observing an indoor area when two people walk into the scene, crossing paths approximately at the frame center, and continue past the camera view. First, the per-pixel algorithm groups similar appearances into segments along the temporal axis as visualized in figure 4.6a which shows a single pixel intensity sequence stacked as a vertical column. The initial temporal segmentation corresponding to the pixel location at the center of the highlighted circle of figure 4.5 would produce temporal groups that would be correlated with the semantic class sequence: (wall appearance, person one appearance, wall appearance, person two appearance, wall appearance). The beginning of each temporal segment defines a point in 3D (2D space-time) called a change
Figure 4.5: Crossing-Paths video. Figures 4.5a through 4.5e are frames from the Crossing-Paths video. Figures 4.5f, 4.5g and 4.5h are a front, top and rear view of the change point vertices in 3-d image space-time. Vertices with matching colors belong to the same change point hull.

Weighted edges, visualized in figure 4.6b as dotted lines, are constructed between the nearest neighboring change vertices and are used to cluster change point vertices by the graph based segmentation algorithm introduced by Felzenszwalb and Huttenlocher [23]. Figures 4.5f through 4.5h show the change point vertices and hulls for the "Crossing-Paths" scene, visualized as points in 3-d image space-time. Points with matching colors represent the position of change point vertices in space-time that have been grouped into the same change point hull. While there may be appearance variations within a single object class (the person walking from right to left has a multi-colored shirt, beige pants and dark hair), the change point hulls exhibit a coherent planar shape. Figure 4.5f shows the change point vertices from the front and figure 4.5h show the coplanar change point vertices created the people occlude the wall over time. The wall appearance is so uniform that looking at
Figure 4.6: Figure 4.6a shows the independent per-pixel temporal segmentation of intensity subsequences into homogeneous groups with similar appearance and duration. The beginning of each sequence, known change point vertices, are shown in fig. 4.6b as black spherical points in 3D (2D image space and time). Each change point vertex is connected to its four nearest neighbors and assigned an edge weight proportional to appearance and duration similarity measures and is used for segmenting the change hull.

If the visualizations in figures 4.5f through 4.5h are difficult to interpret, consider a simplified crossing-paths sequence where one-dimensional objects, moving at (potentially different) constant speeds in opposite directions are imaged by a one-dimensional camera as conceptualized in figure 4.7. The one-dimensional imaging device is shown as the red line and two one-dimensional objects traverse the scene, one traveling from left to right and the other right to left. The change point vertices of the right and left bound object are shown as a function of 2-d space-time as green and blue dots respectively. The positions of the change point vertices in both cases follow a linear path through the 2-d space-time, owing to the constant velocity of the objects. The velocity and direction of each object is proportional the magnitude and sign of the slope of the colinear change point vertices. Figures 4.5f through 4.5h may now be understood as the extension of the simple 1-d example, where a camera projects three-dimensional objects onto a two-dimensional plane for each frame of a video sequence.
Figure 4.7: Crossing paths 1-d example. A pair of 1-dimensional objects traveling across a scene in opposite directions at constant speed are imaged by a one-dimensional camera, visualized by as the red line. The path of change point vertices associated with the object moving from left to right is visualized with green dots. The path of the change point vertices associated with the object moving in the opposite direction is shown as a dotted blue line. The vertices of each object trace a line in 2-d space-time where the magnitude and sign of the line’s slope is proportional to object velocity and direction.

As shown in the "Crossing Paths" experiment, coherent object motion induces planar geometries with respect to the change point hulls while the velocity of object motion is correlated with the normal direction of the change-point hull with respect to the principle axis of the camera. Furthermore, by observing the relative positions in space and time of change point hulls, it is possible to reason about object trajectory, velocity and occlusion while further experiments reveal image noise, random and incoherent motion induce elliptical or spherical change-point surfaces.

An example of a more complex change hull geometry generated by periodic motion is shown in fig. 4.8d. Figures 4.8a, 4.8b, and 4.8c show zoomed in views of one of the wind turbines in different frames of the Fields Point video initially shown in chapter 3. The wind turbine moves in a periodic circular motion which pixel-level change detection algorithms struggle to model as background motion. However, the recovered change point vertices and hulls exhibit a coherent structure with a global helical shape that can be approximated locally with connected planes. The helical pattern and resulting change point hulls are visualized in figure 4.8d where points show the 3D space-time locations of change point vertices and colors denote clustered change point hulls.
Figure 4.8: Visualizing the change point geometry induced by the periodic motion of the wind turbine in the Fields Point video sequence. The rotating turbine blades induce change point vertices with a global helix-like structure in 3D space-time.

The singular values of a 3D spatial covariance matrix, computed by *Singular Value Decomposition* (SVD), composed of the 3D locations of change point vertices within the same change point hull (cluster) is used to quantify the "planarity" of the space-time discontinuity for identifying coherent motion and recognizing meaningful scene changes.
4.3 Constructing Change Hulls via Efficient Graph Segmentation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>A finite collection of graph vertices, {v_1, v_2, \ldots}</td>
</tr>
<tr>
<td>E</td>
<td>Graph edges, a finite collection of pairs of connected vertices, {e_k = (v_i, v_j) \mid v_i, v_j \in V, k \in {1, 2, \ldots}}</td>
</tr>
<tr>
<td>G = {V, E}</td>
<td>An undirected graph, a tuple of vertices V and edge connections E</td>
</tr>
<tr>
<td>(\omega(e))</td>
<td>Edge weight function, a mapping from the set of edges to non-negative reals</td>
</tr>
<tr>
<td>(\mathcal{T})</td>
<td>A spanning tree of an undirected graph G</td>
</tr>
<tr>
<td>(\mathcal{T}_{\text{min}})</td>
<td>The minimum spanning tree of a weighted undirected graph G</td>
</tr>
<tr>
<td>(\mathcal{S})</td>
<td>A segmentation of graph G</td>
</tr>
<tr>
<td>(C)</td>
<td>A component or cluster of a segmentation (\mathcal{S})</td>
</tr>
<tr>
<td>Int ((C))</td>
<td>Internal similarity measure of a component</td>
</tr>
<tr>
<td>(\Delta(C_1, C_2))</td>
<td>External similarity measure between two components</td>
</tr>
<tr>
<td>(\tau(C))</td>
<td>A function mapping from components to non-negative reals</td>
</tr>
<tr>
<td>(\mathcal{D}(C_1, C_2))</td>
<td>A binary predicate taking value one if there is evidence of a boundary between two components and zero otherwise</td>
</tr>
</tbody>
</table>

Table 4.4: Summary of notation of summary used for constructing change point hulls.

Grouping change point vertices to form change point hulls is posed as a graph segmentation problem which is solved using the algorithm presented by Felzenszwalb and Huttenlocher in [23] and reviewed in this section. A weighted undirected graph is a tuple of sets, \(G = \{V, E\}\) augmented with a weight function \(\omega(e)\) such that \(e \in E\). \(V\) is a set of vertices, atomic elements of the graph and \(E\) is a set of edges which connect pairs of vertices

\[ E = \{e_k = (v_i, v_j) \mid v_i, v_j \in V\} \]  

(4.18)

If a graph is undirected, then the edges are unordered pairs of vertices, e.g., for vertices \(u\) and \(v\), \((u, v)\) and \((v, u)\) are equivalent edges or connections.

A segmentation \(\mathcal{S}\) of a graph \(G\) is a partition of vertices into a set of disjoint components

\[ \mathcal{S} = \{C_i \mid C_i \subseteq V, C_i \cap C_j = \emptyset \forall i \neq j, \text{ and } i, j \in \{1, 2, \ldots\}\} \]  

(4.19)

where \(C\) is a connected component in the original graph \(G\). A connected component of \(G\) is a subset of vertices that are connected by edges. In other words, a segmentation of a graph \(G = \{V, E\}\) can
be thought of as a function which produces a second graph $G' = \{V, E'\}$ with the same vertex set as $G$ but potentially, some edges have been removed implying $E' \subseteq E$. Vertices in a component of the segmentation retain their edges while edges between vertices in different components are deleted.

In the context of change point hull construction, vertices of the graph are defined in terms of the locations of change point vertices and edges connect the eight nearest neighbors of each vertex. A segmentation of the resulting graph is a collection of disjoint subsets of change point vertices that have been grouped together. Change point hulls are thus the collection of vertices that are members of the same component after segmentation.

While there are many graph clustering algorithms in the standard literature, the algorithm of Felzenswalb and Huttenlocker was chosen because it may be implemented by a simple, iterative procedure that scales nearly linearly with the number of edges in the original graph while producing segmentation solutions with theoretical guarantees concerning desirable global properties. Specifically, the algorithm proposed in [23] is designed to satisfy the following principles:

**Principle 3.** Vertices in a common component should be similar.

**Principle 4.** Vertices in different components should be dissimilar.

The similarity of two vertices is expressed via edge weights, a function which maps from the set of graph edges to non-negative reals.

$$\omega \left( (v_1, v_2) \right) : E \rightarrow \mathbb{R}^+$$

The two principles then correspond to the notions that vertices within the same component should have relatively low weights compared to edges weights between vertices in different components. While the graph segmentation algorithm is developed in terms of a general weight function, the similarity function of equation 4.17 is used to compute edge weights for constructing change point hulls and can be evaluated in constant time via a state-mean look up table and simple arithmetic.

To compute a segmentation that satisfies principles 3 and 4, a binary predicate is defined that is mapping between neighboring components and \{0, 1\}. A one indicates there is evidence of a
boundary between vertices connecting the two components and a zero does not. The binary predicate \( D(C_1, C_2) \) is defined as a threshold function of the weight differences internal each of the two components compared against the minimum edge weight between the components. The internal component difference is defined as

\[
\text{Int}(C) = \max_{e \in T_{\text{min}}(C,E)} w(e)
\]

(4.21)

where \( T_{\text{min}}(C,E) \) is the minimum spanning tree of the vertices in component \( C \) that have been connected according to the original graph. For convenience, \( T_{\text{min}}(C,E) \) will often be refereed to as \( T_{\text{min}} \) when the context is clear.

**Minimum Spanning Trees** A spanning tree for a weighted undirected graph \( G = \{V, E\} \) is also a graph \( T = \{V, E'\} \) composed of the vertices of \( G \) and a subset of edges from \( G, E' \subseteq E \), that create a path traversing all vertices of \( G \) without cycles. A cycle is a path in a graph, a sequence of non-repeating edges, connecting any vertex \( v \) back to itself. Simple examples of a graph without cycles and a graph with cycles is shown in figure 4.9. In general, there exists a collection of spanning trees for a connected graph \( G \). The minimum spanning tree \( T_{\text{min}}(V,E) \) is a spanning tree of \( G = (V, E) \) that minimizes the sum of all edge weights with respect to the collection of all spanning trees of \( G \).

![Graphs](image)

(a) Example of an undirected graph without cycles.  
(b) Example of an undirected graph with a cycle.

Figure 4.9: Examples of a graph without cycles and a graph with a cycle.

A simple example of computing a minimum spanning tree is shown in figure 4.10. Figure 4.10a
visualizes an arbitrary weighted undirected graph with vertices, edges and weight function,

\[
G = \{V = \{a, b, c, d, e\}, E = \{e_1 = (a, b), e_2 = (a, e), e_3 = (b, d), e_4 = (d, c), e_5 = (e, c)\}\}
\]

\[
W = \{\omega(e_1) = 7, \omega(e_2) = 3, \omega(e_3) = 1, \omega(e_4) = 5, \omega(e_5) = 2\}
\]

where \(W\) is the collection of all edge weights. Figures 4.10b, 4.10c and 4.10d enumerate the possible spanning trees for \(G\), denoted \(T_1\), \(T_1\) and \(T_3\). The minimum spanning tree is the spanning tree with the minimal sum of edge weights with respect to all possible spanning trees. In this example, the minimum spanning tree \(T_{\text{min}}\) may be computed as

\[
T_{\text{min}} = \arg \min_{T \in \{T_1, T_2, T_3\}} \left\{ \sum_{e \in T} \omega(e) \right\}
\]

which, for the example, is \(T_{\text{min}} = T_1\). Algorithms for computing minimum spanning trees were introduced by Kruskal [78] and Prim [79] which propose iterative growing algorithms that converge to the exact minimum spanning tree for general graphs in \(O(|E| \log(|V|))\) and \(O(|E| + |V| \log(|V|))\) time respectively. Due to the algorithmic and asymptotic complexity similarities between Kruskal’s and Felzenszwalb’s algorithms, a brief review of Kruskal’s algorithm for computing minimum spanning trees is presented.
(a) Example of a weighted undirected graph to illustrate computing a minimum spanning tree.

(b) $T_1$, the first possible spanning tree for the graph shown in figure 4.10a. The sum of all edge weights for this tree is 11.

(c) $T_2$, the second possible spanning tree for the graph shown in figure 4.10a. The sum of all edge weights for this spanning tree is 15.

(d) $T_3$, the third possible spanning tree for the graph shown in figure 4.10a. The sum of all edge weights for this spanning tree is 13.

Figure 4.10: Simple example of computing the minimum spanning tree for a weighted undirected graph. Figure 4.10a shows a particular graph for which the minimum spanning tree is to be computed. There are three possible spanning trees for the graph, $T_1$, $T_2$, and $T_3$, shown in the figures 4.10b, 4.10c and 4.10d respectively. The sum of the edge weights for spanning tree $T_1$ is eleven, the sum of edge weights for $T_2$ is fifteen and the sum of edge weights for $T_3$ is twelve. Therefore, $T_1$ is the minimum spanning tree.
Kruskal’s Algorithm for Computing Minimum Spanning Trees

Kruskal’s algorithm for computing the minimum spanning tree $T_{\text{min}}$ of a graph $G = \{V, E\}$ is defined in terms of set operations on the edges and vertices of the original graph [80]. The set of edges representing the minimum spanning tree is initialized to the empty set and each vertex is considered as a member of its own set, a collection of $|V|$ single-element sets. The final initialization step is to sort the edges of $G$ in non-decreasing order with respect to edge weight. Iterating over each edge in the sorted list, $e_k = \{v_i, v_j\}$, if the vertices of the current edge are not members of the same set, then the edge is added to the minimum spanning tree graph and the sets that contain $v_i$ and $v_j$ are merged together.

Kruskal’s algorithm requires three set operations: an operation to create sets, an operation to find the set id of a vertex, and an operation to merge two sets. Following the conventions of [80], these operations are denoted $\text{MAKE-SET}(v)$, $\text{FIND-SET}(v)$, $\text{UNION}(v_i, v_j)$ for vertices $v, v_i$, and $v_j$. For completeness, pseudo code based on the exposition of [80] for Kruskal’s algorithm given in listing 5.

The run-time complexity of Kruskal’s algorithm, which is closely related to the clustering algorithm used for building change point hulls, is dependent on the efficiency of the set operations which will be analyzed in a later section. Note that it is possible to prove Kruskal’s algorithm will converge to the minimum spanning tree for any solution which sorts edges by weight in non-decreasing order regardless of how edges with identical weight are handled.

Algorithm 5 Kruskal’s Minimum Spanning Tree Algorithm

1: function KRUSKAL($G = \{V, E\}, \omega$)
2: $T_{\text{min}} \leftarrow \emptyset$
3: for all $v_i \in V$ do
4: \hspace{1em} MAKE-SET($v_i$)
5: end for
6: Sort all edges in $E$ by $\omega$
7: for all $e = (v_i, v_j) \in E$ do
8: \hspace{1em} if $\text{FIND-SET}(v_i) \not= \text{FIND-SET}(v_j)$ then
9: \hspace{2em} $T_{\text{min}} \leftarrow \{T_{\text{min}} \cup e_k\}$
10: \hspace{2em} UNION($v_i, v_j$)
11: end if
12: end for
13: return $T_{\text{min}}$
14: end function
Figure 4.11: Computing the minimum spanning tree of the example graph in figure 4.10. The list of edges sorted by weight is \( \{e_3, e_5, e_2, e_4, e_5\} \). Kruskal’s algorithm iterates over the sorted edge list iteratively growing the minimum spanning tree. The first four iterations of Kruskal’s algorithm are visualized in figures 4.11a, 4.11b, 4.11c and 4.11d where the highlighted edges represent the members of \( T^{\text{min}} \) as it is updated in each step. The solution \( T^{\text{min}} = T_1 \) is the same as when computed by brute force enumeration by inspection in figure 4.10.

Figure 4.11 visualizes the iterations of Kruskal’s algorithm for computing the minimum spanning tree of the graph shown in figure 4.10a. The edges of the graph are sorted in non-decreasing order with respect to the edge weight function and the algorithm is initialized to contain sets consisting of each individual vertex. With reference to the weighted edges as enumerated in equation 4.22, the list of sorted edges is \( E = \{e_3, e_5, e_2, e_4, e_5\} \). Iteration one tests if the vertices of the edge \( e_3 = (b, d) \) are in the same set. They are not so \( e_3 \) is added to the minimum spanning tree, as indicated by the highlighted edge in figure 4.11a, and the singleton sets containing vertices \( b \) and \( d \) are merged by a set union operation. Continuing for the rest of the iterations yields the solution visualized in figure 4.11d where the minimum spanning tree consists of all the vertices and the highlighted
subset of edges. The minimum spanning tree returned by Kruskal’s algorithm is the same as was
determined by inspection in figure 4.10b.

Returning to the discussion of the graph segmentation algorithm, the difference between compo-
nents is measured as the minimal edge weight connecting vertices that traverse the components and
may be expressed as [23],

\[ \Delta (C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2} \omega ((v_i, v_j)) \] (4.24)

Two components, \( C_1 \) and \( C_2 \) are considered to be neighbors if there exists at least one edge from
a vertex in \( C_1 \) to any vertex in \( C_2 \). For any two components that are not neighbors, the external
component difference is by definition infinite: \( \Delta (C_1, C_2) = \infty \).

Having defined the internal and external difference of and between components, the binary
evidence predicate may be computed according to,

\[
D(C_1, C_2) = \begin{cases} 
1 & \text{if } \Delta (C_1, C_2) > MInt (C_1, C_2) \\
0 & \text{otherwise}
\end{cases}
\] (4.25)

A predicate value of one indicates there is evidence of a boundary between components whereas a
predicate value of zero indicates no evidence of a boundary. The \( MInt \) function is called the minimal
internal difference and is defined as [23],

\[
MInt = \min \{ Int (C_1) + \tau (C_1), Int (C_2) + \tau (C_2) \} \] (4.26)

\( \tau (C) \) is a function mapping from components to the non-negative reals. The purpose of the \( \tau \) term is
to regularize against favoring small components. It is very likely that a large component will exhibit
a minimum spanning tree with larger weight than a smaller component. Therefore, the function
\( \tau (C) \) is inversely proportional to component size, defined as the number of vertices the component
contains and denoted \( |C| \),

\[
\tau (C) = \frac{k}{|C|} \] (4.27)
is a constant parameter that defines an *apriori* natural scale of components and is set to 300 in all experiments.

To gain intuition about the influence of each term on the binary predicate, realize there are only two possibilities for the minimal internal difference measure, either

$$\text{Int}(C_1) + \tau(C_1) < \text{Int}(C_2) + \tau(C_2)$$  \hspace{1cm} (4.28)

is true or its converse is. Arbitrarily, assume 4.28 is true. Then the evidential predicate is true only if

$$\Delta(C_1, C_2) > \text{Int}(C_1) + \tau(C_1)$$  \hspace{1cm} (4.29)

This implies there is evidence of a boundary only if,

$$\Delta(C_1, C_2) - \text{Int}(C_1) > \tau(C_1)$$  \hspace{1cm} (4.30)

Equation 4.30 makes it easier to see that there is evidence of a boundary between components only if the difference between the inter-component similarity and the minimal internal similarity measure is greater than a threshold that scales relative to the size of the components.

The segmentation algorithm proposed by Felzenszwalb and Huttenlocher is given by listing 6. The algorithm is similar to that proposed by Kruskal and begins by sorting the edges in non-decreasing order with respect to edge weight. The segmentation solution is then initialized to a collection of singleton sets where each vertex is its own set. The list of sorted edges is then iterated where the variables $C_1$ and $C_2$ denote the state ids associated with the vertices contained in the edge. In the pseudo code of listing 6, the binary arguments to the FIND-SET function explicitly show the dependence of the find set operation on the current state of the segmentation. Logically, a merge is only considered if the vertices of the current edge are contained in different sets. If the edge weight connecting the two vertices is less than the minimum distance between the two components, the components are merged otherwise the algorithm proceeds to the next edge. By inspecting listing 6, the complexity of the algorithm is the sum of the edge sort which may be solved in the general case.
in $O(|E| \log (|E|))$ time and the complexity of each iteration multiplied by the number of edges. In each iteration, there are a minimum of two set-find operations to determine if the vertices belong to the same set given the current segmentation. If the vertex sets are disjoint, the minimum internal distance measure must be evaluated and, potentially, a set-union is performed.

Algorithm 6 Graph segmentation algorithm of Felzenszwalb and Huttenlocher

1: function Graph-Segmentation($G = \{V, E\}, \omega$)
2: Sort $E$ by non-decreasing edge weight
3: $S \leftarrow \emptyset$
4: for all $v \in V$ do
5:     $S \leftarrow \{S \cup \text{MAKE-SET}(v)\}$
6: end for
7: for all $e = (u, v) \in E$ do
8:     $C_1 \leftarrow \text{FIND-SET}(S, u)$
9:     $C_2 \leftarrow \text{FIND-SET}(S, v)$
10:    if $C_1 \neq C_2$ then
11:        if $\omega(e) \leq MInt(C_1, C_2)$ then
12:            \text{UNION}(C_1, C_2)
13:        end if
14:    end if
15: end for
16: return $S$
17: end function

Surprisingly, evaluating $MInt(C_1, C_2)$ can be accomplished trivially in constant time. Consider the edge $e_k = (u, v)$ such that $u, v \in v$. Furthermore, let $C_1$ and $C_2$ be the sets containing $u$ and $v$ respectively and with $C_1 \neq C_2$. Due to the initial edge sorting step the weight of the edge in the current iteration is less than or equal to the weight of the edge in the next iteration: $\omega(e_k) \leq \omega(e_{k+1})$. Therefore, the edge $e_k$ is the edge with minimal weight connecting the components $C_1$ and $C_2$. If the components $C_1$ and $C_2$ are singletons, containing only $u$ and $v$ respectively, if it is decided the components should be merged, the result is a minimum spanning tree of a subset of graph $G$. Because the edge with minimum weight is used to merge the two components, at any point of the iteration, the merge will result in a new, larger component that is also a minimum spanning tree of the vertices contained in $C_1$ and $C_2$. The weight of the merged set will then by definition be

$$Int(C_1) + Int(C_2) + \omega(e_k) \quad (4.31)$$
which can be computed and cached with each iteration for computing the minimum internal distance in constant time at any step of the iteration. Since $\text{MInt}(C_1, C_2)$ may be computed in constant time, the complexity of the Felzenszwalb segmentation algorithm is asymptotically identical to Kruskal’s algorithm and both are dependent on the implementation of the set operations.

It is proved in [23] that the algorithm in listing 6 produces a segmentation solution that is neither too coarse or too fine with respect to the evidence predicate. The concept that a segmentation may be too coarse or too fine are made rigorous by the following definitions,

**Definition 1.** A segmentation $S$ is too fine if there is some pair of regions $C_1, C_2 \in S$ without evidence of a boundary between them with respect to the evidence criteria $D(C_1, C_2)$.

**Definition 2.** A segmentation $S$ is too coarse if there exists a proper refinement of $S$ that is not too fine.

To understand definition 2, it is necessary to define a refinement of a segmentation. Let $S$ and $S'$ be two segmentations of a set. $S'$ is a refinement of $S$ if each component of $S'$ is contained within a component of $S$. A proper refinement is a non-trivial refinement, $S' \neq S$ [23]. It is noted there always exists a segmentation that is not too coarse or too fine which may not be unique in general. It is shown that although the graph segmentation algorithm is a local iterative approach, the result satisfies the global properties in definitions 1 and 2.

To finish the analysis of the change point vertex clustering algorithm, it is necessary to understand the data structure used for maintaining set membership and the algorithms for performing the Make-Set, Find-Set, and Union operations. [23] recommends a disjoint-set forest with union by rank and path compression as presented in [80] and the implantation used to construct change point hulls follows suit.
Figure 4.12: Visualizing disjoint set forests. Component sets are represented as disjoint trees and a segmentation is the set of all disjoint trees. Elements of the tree point to their parents and the root of the tree points to itself. In this example there are three component sets identified by their roots, $C_1 = a$, $C_2 = e$, $C_3 = f$. $C_1$ contains the elements \{a, b, c, d\}, $C_2$ contains the single element \{e\}, and component $C_3$ contains \{f, g, h\}.

### 4.3.1 Disjoint-Set Forests

So far, it has been shown that the run-time of the graph segmentation algorithm depends on the run time of the \texttt{MAKE-SET}, \texttt{FIND-SET} and \texttt{UNION} operations. As recommended by [23] and detailed in [80] a collection of disjoint sets is represented as a collection of rooted trees. Every node in the tree points to its parent and the tree root points to itself. The \texttt{MAKE-SET} operation creates a tree with one element which points to itself. The \texttt{FIND-SET} operation is augmented with a path compression heuristic to limit the operations run-time and the \texttt{UNION} uses a rank-union heuristic also to limit the operational complexity.

**Rank-Union** When performing a set union to merge two disjoint sets into one, it is more efficient for future calls to the \texttt{FIND-SET} operation if the root of the smaller set is added as a child to the root of the larger tree. Tree size is taken to mean the height of the tree which is the longest path from the root to a tree leaf. Rather than computing the exact tree height, a rank-union heuristic is introduced that saves computation by approximating tree height by an upper bound. Each tree is associated with a rank. If the ranks of two sets that are passed to the \texttt{UNION} function are not equal, the set with smaller rank is added to the tree with larger rank and the rank of the union result is the rank of the larger set. If the ranks of two sets to be merged are equal, the sets are joined arbitrarily and the rank is incremented. When a \texttt{MAKE-SET} operation creates a new tree, the rank
of the singleton tree is initialized to zero.

**Path Compression** Path compression is another heuristic for making the run time of set operations more efficient when using the disjoint-set forest representation. Every time a \( \text{Find-Set}(v) \) operation is performed on an element, the parent of every parent on the path connecting \( v \) to the root is connected directly to the root via a recursive procedure prior to returning the set representative. This increases the likelihood that a vertex contained in any set will have a direct connection to the root making the find operation much cheaper.

Listings 7, 8 and 9 present pseudo-code for the \( \text{Make-Set} \), \( \text{Find-Set} \) and \( \text{Union} \) operations inspired by [80]. Note the arguments of these functions, such as \( v \), are assumed to be abstract data types with member variables \( v.\text{rank} \) and \( v.\text{p} \) that represent the rank and pointer to the parent of element \( v \).

It is shown in [80] that performing \( n \) operations on \( m \) elements represented using the disjoint-forest set data structure combined with rank-union and path compression heuristics results in an asymptotic run-time complexity of \( O(mA(n)) \) where \( A(n) \) is the inverse Ackerman function. For all practical values of \( n \), the inverse Ackerman function is less than equal to the constant four [80]. Hence, by using a disjoint-set forest representation, the performance of the main loops in Kruskals and Felzenszwalb’s algorithms are essentially linear in the number of edges.

**Algorithm 7** Pseudo-code for the \( \text{Make-Set} \) operation using a disjoint-set forest

```plaintext
1: function \( \text{Make-Set}(v) \)
2: \( v.\text{p} \leftarrow v \)
3: \( v.\text{rank} \leftarrow 0 \)
4: end function
```
Algorithm 8 Pseudo-code for the FIND-SET operation with path compression

1: function FIND-SET(v)
2:   if v.p ≠ v then
3:     v.p ← FIND-SET(v.p)
4:   end if
5:   return v.p
6: end function

Algorithm 9 Pseudo-code for the UNION operation with rank-union

1: function UNION(u, v)
2:   C₁ ← FIND-SET(u)
3:   C₂ ← FIND-SET(v)
4:   if C₁.rank > C₂.rank then
5:     C₂.p ← C₁
6:   else
7:     C₁.p ← C₂
8:     if C₁.rank = C₂.rank then
9:       C₂.rank ← C₂.rank + 1
10:   end if
11: end if
12: end function

4.3.2 Performance Analysis of Change Point Hull Construction

<table>
<thead>
<tr>
<th>Video</th>
<th>Resolution</th>
<th># Vertices</th>
<th># Edges</th>
<th># Hulls</th>
<th>Run Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop-Forward</td>
<td>312 × 328 × 275</td>
<td>450,062</td>
<td>1,343,052</td>
<td>10,872</td>
<td>370</td>
</tr>
<tr>
<td>Walk-Forward</td>
<td>312 × 328 × 200</td>
<td>43,1136</td>
<td>1,281,951</td>
<td>9,145</td>
<td>490</td>
</tr>
<tr>
<td>Crossing-Paths</td>
<td>312 × 328 × 115</td>
<td>749,280</td>
<td>2,296,544</td>
<td>19,142</td>
<td>590</td>
</tr>
<tr>
<td>Fields Point</td>
<td>720 × 480 × 1000</td>
<td>2,714,972</td>
<td>8,871,314</td>
<td>11,703</td>
<td>2,700</td>
</tr>
<tr>
<td>Swing</td>
<td>720 × 480 × 2400</td>
<td>14,922,356</td>
<td>49,458,643</td>
<td>470,863</td>
<td>16,500</td>
</tr>
</tbody>
</table>

Table 4.5: Run-time performance for building change point hulls. The resolution of each video, the number of change point vertices, number of edges, number of resulting change point hulls after segmentation and the run time in milliseconds of the graph segmentation algorithm is shown for each experiment.

Table 4.5 shows algorithmic performance statistics for the change point clustering algorithm using the graph segmentation algorithm of Felzenszwalb and Huttenlocher [23]. The Crossing-Paths and Stop-Forward video sequences are relatively shorter and have less per-frame resolution than the other experiments and the graph segmentation algorithm takes less than a second to segment one and two million edges into ten and twenty thousand change point hulls for these videos respectively.
The Fields Point video contains over three hundred and forty five million intensity values considering the resolution of each frame and number of frames. After the DDHMM segmentation there are approximately three million change point vertices which are connected by almost nine million edges prior to segmentation. The segmentation algorithm takes only three seconds to produce a change point hull segmentation with close to twelve thousand change point hulls.

The Swing sequence is the largest experiment with 2,400 frames and $720 \times 480$ pixels per frame, the video sequence is composed of over eighty two million intensity values. The real-time DDHMM algorithm partitions the intensity values into approximately fifteen million intensity sequences and after connecting change point vertices to their eight nearest neighbors, there are almost fifty million edges. Yet, even with this massive graph, the segmentation algorithm produces nearly half a million change point hulls in approximately seventeen minutes.

Note that in each video, the change point hull representation offers a huge compression ratio compared to the full resolution of the video sequence. As will be shown later, the change point hulls are semantically important patches of video which contain information about moving structures. Therefore, the change point hull representation saves enormous amounts of processing power by summarizing important aspects of video sequences with a smaller amount of objects.

![Figure 4.13](image.png)

Figure 4.13: Run-time performance building change point hulls using the graph segmentation algorithm of Felzenszwalb and Huttenlocher [23]. As seen in the figure, the algorithm for grouping change point vertices scales linearly with the number of edges between change point vertices as predicted by the theoretical analysis.
Figure 4.13 plots the run time of the segmentation algorithm for building change point hulls versus the number of edges created in each data set. The graph shows the run-time performance is almost perfectly linear in the number of edges as predicted by the theoretical analysis.

**Black-Square video** As an initial proof of concept, a series of synthetic video sequences were generated to investigate and aid in the development of change point geometry. Figure 4.14 contains the analysis of a synthetic sequence of images containing a small black square traversing a white background and is referred to as the Black-Square video. This simple experiment isolates the effects of the DDHMM learning algorithm as the simple intensity patterns guarantees a perfect per-pixel intensity segmentation and allows the analysis to focus on the change point vertices and hulls.

Figures 4.14a through 4.14d show example frames extracted from the video sequence. Figures 4.14e, 4.14f, and 4.14g visualize the change point vertices and hulls constructed by the algorithm proposed in section 4.2 in 3-d image space-time. Note that due to the density of change vertices in some video sequences, the change hulls appear to be surfaces. However, change point vertices and hulls are actually a dense collection of 3D points in space-time. The point nature of change point vertices will be more clearly seen in some visualizations than others, depending on the resolution of the video sequence and the complexity of the appearance dynamics in the scene.

When the DDHMM algorithm begins with the first frame of video, a state is created at each pixel for the first intensity seen in the frame. The collection of vertices associated with the first frame is visualized as the set of purple coplanar points in figures 4.14e and 4.14f. These initial coplanar points simultaneously serve as a reference for the camera plane as each initial change point vertex corresponds to a pixel location in the first frame of video.

The change point vertices and hulls corresponding to the moving black square are also a set of coplanar points, and imply a "ramp" structure; a plane lying at an angle with respect to the vertices on the camera plane. The angle of the ramp compared to the camera plane is proportional to the velocity of the moving square.
Figure 4.14g shows a straight on view of the ramp corresponding to the change hulls associated with the black square, where the initial coplanar plane vertices lying on the camera plane have been removed from view for visualization. Vertices with the same colors are members of the same change point hull. While there are some small change point hulls that have incorrectly been grouped outside the main change point hull set, these are small sets of vertices with irregular shapes that will be filtered out with additional analysis. The mistakes are due to interpolation performed at the edges of the black square and white background when generating the synthetic image. The edges of the black square against the white background result in a blurred gray color and subsequently a few change vertices go through a DDHMM state that is associated with a gray color before transitioning to a state representing a black or white intensity. These gray states are logically grouped together by the graph segmentation algorithm.

Figure 4.14f shows a top down view of the change point vertices and hulls. Note there are two main change point hull components, the purple set of vertices and blue vertices. Additionally, these change point hull components are associated with three sets of coplanar points 3-d image space-time: The coplanar purple points lying on the camera plane, the coplanar blue points lying on the ramp corresponding to the black moving square, and an additional set of coplanar purple points directly behind the set of coplanar blue points. The latter purple set of points represents change point vertices that are created as the black square moves forward and the background is revealed after being occluded by the black moving object.

**Black-Square-Occlusion video** Figure 4.15 is an additional synthetic image sequence generated to investigate the effects of occlusion on change point geometry and is referred to as the Black-Square-Occlusion video. The scene again features a black square moving left to right across a white background with constant velocity. However, there is a gray rectangle in the center occluding the black square with respect to the camera. Figures 4.15a, 4.15b, 4.15c, and 4.15d show example frames taken from the Black-Square-Occlusion video sequence. Figure 4.15c shows the black square passing
Figure 4.14: *Black-Square* video sequence. Figures 4.20a, 4.20b, 4.20c and 4.20d show example frames taken from the Black-Square video sequence which is a synthetically generated scene consisting of a black square traversing a white background. Figures 4.14e and 4.14f show a front and top view of the change vertices and hulls respectively. Figure 4.14g shows a straight on view of the change vertices and hulls of the moving object after removing the initial coplanar vertices lying on the spatio-temporal positions of the first frame. By inspection of the change point vertices and hulls, it may be seen that an object traversing the camera plane with constant velocity will induce a collection of coplanar change point vertices at an angle with respect to the camera plane.

behind the gray barrier and figures 4.15e, 4.15f, and 4.15g visualize the change point vertices and hulls in 3D image space-time.

Again there is a set of coplanar points lying on the camera plane each created by the per-pixel DDHMM algorithm after observing the first frame of video. In contrast with the Black-Square video,
in this example the points lying on the camera plane are segmented into two change point hulls as visualized by the coplanar green and blue points as seen in figure 4.15e. The green points correspond to the white background that are clustered together by the graph segmentation algorithm. Pixel locations corresponding to the occluding gray rectangle at the center of the first frame of video are grouped in a separate component visualized as the blue coplanar change point vertices lying on the camera plane.

Figure 4.15f shows a top view of the change point vertices and hulls. As expected, the square traversing the frame creates a set of coplanar change point vertices and hulls at an angle with respect to the camera frame. However, the occlusion breaks the ramp-like shape corresponding to the moving black square into two disjoint change hull components. This is a natural consequence of occlusion effecting change point geometry. There are no visible discontinuities from the vantage point of the camera as the moving object passes behind an object with static appearance. Due to the constant velocity of the black square, the two planes corresponding to object movement are coplanar and maintain the same constant angle with respect to the camera plane.

Figure 4.15g shows the change point vertices and hulls associated with the moving square from the front. The vertices on the camera plane removed for clarity. Due to the occluding rectangle in the center of the frame, the change point vertices corresponding to the black square are not connected by weighted edges when the graph is constructed in the hull segmentation step. Therefore, while the vertices corresponding to the black square are clustered into a single dominant component in figure 4.14g, the occluding gray square causes a gap which divides the coplanar vertices associated with the moving square into two dominant components visualized as vertices colored as light blue and purple.

**Synthetic-Car-Down and Synthetic-Car-Down-Noise videos**

the Synthetic-Car-Down and Synthetic-Car-Down-Noise videos shown in figures 4.16 and 4.17 are experiments consisting of the same synthetic scene with uniform noise on the interval $[-20, 20]$ added to each pixel in the latter
Figure 4.15: Black-Square-Occlusion video sequence. Figures 4.15a through 4.15d show example frames taken from the Black-Square-Occlusion sequence which consists of a black square moving from left to right across a white background. The moving black square is occluded at the frame center by a gray rectangle. Figures 4.15e, 4.15f, and 4.15g visualize the resulting change point vertices and hulls in 3-d image space-time. The change point geometry consists of two disjoint sets of coplanar change point vertices lying at an angle with respect to the camera frame. These sets of coplanar points correspond with the change vertices created to account for the moving black square. The scene was designed to replicate, however simplistically, an idealized car traveling down a road. Due to the lower resolution of these videos, it is easier to see in figures 4.16e and 4.16f that change point vertices are points in 3-d image space-time and change point hulls are collections of
change vertices as shown by vertices with matching colors. Again, there exists a set of coplanar vertices lying on the camera plane due to the initialization of the DDHMM learning module. The graph segmentation algorithm creates three change point hull groups on the camera plane. One component for the gray strip, and two components associated with the white pixels on the left and right of the gray strip. The moving black rectangle creates another set of coplanar change vertices at an angle with the camera’s principal axis, shown in orange. On top of the orange change vertices is another set of vertices lying on a plane with the same slope as the orange vertices and correspond to the appearance discontinuities associated with the black car moving forward to reveal the gray background. These vertices have been grouped together with the vertices that account for the initial observation of the gray strip of road and are therefore colored blue as well.

Figures 4.17e and 4.17f show that the DDHMM assignment algorithm is able to accurately model much of the additive noise and reproduce the vast majority of change vertices that are present in the experiment without noise. While there are some errant change vertices, they are clustered together into small groups with irregular shapes and are easily filtered out in a later step.

**Synth-Spin video**  Figures 4.18a through 4.18h are examples of frames taken of a synthetic video of a rotating black line on a white background. The purpose of this experiment is to investigate the effect of periodic appearance dynamics on change point geometry in a controlled environment. Like the turbine in figure 4.8, the resulting change point vertices and hulls create a spiral staircase or helix shape in 3-d image space time.

**Walk-Forward video**  The experiment shown in figure 4.19 shows a real-world scene consisting of a pedestrian traveling from right to left across the camera frame with approximately constant velocity. The collection of change point vertices, prior to hull construction, are shown as points with uniform color in figure 4.19d. A front, side and top view of the change point vertices colored according to hull membership are shown in figures 4.19e, 4.19f and 4.19g. The change point hull components are much smaller compared to the synthetic cases where the appearance of the moving object was
Figure 4.16: *Synthetic-Car-Down* video sequence. This is a synthetic scene of a black square traveling down a gray strip surrounded on both sides by white pixels. Figures 4.16a through 4.16c are example frames taken from the original video. Figures 4.16e and 4.16f visualize the change point vertices and hulls in 3-d image space-time that are created to account for the scene’s appearance dynamics.

relatively homogeneous. However, the pedestrian contains much more complex appearance patterns than the black square in the synthetic videos and simultaneously moves up and down while walking across the frame. However, the change vertices are still coplanar at an angle with respect to the principal axis of the camera just as in the synthetic examples.
Figure 4.17: *Synthetic-Car-Down-Noise* video sequence. This video sequence is the same as in figure 4.16 except uniform noise on the interval \([-20, 20]\) has been added to every gray value as seen in the example frames in figures 4.17a, 4.17b, 4.17c, and 4.17d. The DDHMM is able to account for the additional pixel-level intensity variance but there are some mistakes. However, these mistakes will be accounted for in a later section when change point hulls are filtered by size (number of member vertices) and shape.

**Stop-Forward video**  Figures 4.20a through 4.20d are frames taken from the Stop-Forward video sequence. The video contains a pedestrian walking from left to right across the frame traveling with constant velocity before slowing to a halt, pausing for a few seconds in the center of the frame, and
Figure 4.18: *Synth-Spin* video sequence. Figures 4.18a through 4.18h show example frames of the Synth-Spin video sequence consisting of a black line rotating counter-clockwise against a white background. The resulting change point geometry shown in figures 4.18i and 4.18j from two vantage points and mirrors a spiral staircase or helix in 3-d image space-time.

continuing out of the camera view to the right, accelerating to an approximately constant velocity.

The global shape of the change points and hulls are seen in figure 4.20e and 4.20f. The movement of the pedestrian creates change point vertices and hulls that trace a 2-d "S"-shaped surface through 3-d image space-time.

Again, if figures 4.20e and 4.20f are difficult to interpret, conceptualize a one-dimensional object
moving into a scene at constant velocity from the left, slowing to a halt and pausing for a few seconds before accelerating to a constant velocity and exiting the scene to the right. Figure 4.21 visualizes the sequence of the change point vertices produced by such an object in 2-d space-time. When the object enters the scene at a constant velocity, the change point vertices trace a line as verified in previous experiments. When the object slows, the line curves upwards as less space is traversed per unit time. When the object pauses, the duration of the last change point vertex is extended until the object begins to move again and a new change point vertex is created. As the object begins to accelerate, the sequence of change point vertices curves such that more space is traversed per unit time. Once the object reaches a constant velocity, the change point vertices follow a linear path until the object leaves the scene. Extending logic of the theoretical one-dimensional example to the three-dimensional case, it is easy to see the same geometric patterns traced by the sequence of change point vertices and hulls in 3-d space-time as shown in figure 4.20e.

**Checker-Back video** Frames from the Checker-Back video are shown in figures 4.22a through 4.22d. The scene features a metallic checkerboard pattern on a wall in the background. The material of the checkerboard creates specular reflections and shadows which make interpreting visual information in this scene more difficult. A pedestrian walks into the scene from left to right, slowing to a stop near the frame center and pausing for a few seconds. The pedestrian then backs out of the scene the way he came. The resulting change point hulls and vertices are shown from different viewpoints in 3-d image space-time in figures 4.22e through 4.22h.

There are a larger amount of change point vertices scattered randomly in 3-d space-time owing to erroneous subsequences created by the DDHMM algorithm due to specular reflections off the metallic background. However, the shapes implied by the change point hulls associated with the errant vertices are either linear or elliptical whereas groups of vertices associated with object movement are locally coplanar. Therefore, these change-point hulls can be removed after further shape analysis.

The global shape of the change point vertices similar to a "U" which can be explained by the
conceptualized 1-d example visualized in figure 4.23. In the 1-d example, a 1-d object follows the movement of the pedestrian in the original Checker-Back video. The object enters the scene from left to right at constant speed, slows to a stop in the center, and exits the way it came after accelerating to a constant speed. As usual, when the object enters the scene with constant speed, the path of the change point vertices trace a line in 2-d space-time until the object slows, at which point the change point vertices trace an upwards curve as the object traverses less space per unit time. No change point vertices are created while the object is at rest, creating a gap between the times the object stopped and started to move again. When the object begins to accelerate from right to left, the change point vertices trace a curve in space-time until a constant velocity is reached and the change point vertices follow a linear trajectory.

Returning to figure 4.22, the change point vertices associated with the pedestrian follow a pattern similar to the conceptual example of figure 4.23. A set of coplanar change point vertices are associated with the pedestrian traversing the scene from left to right at a constant velocity. As the pedestrian slows, the resulting change point vertices are associated with a curved surface until the pedestrian stops. Although the pedestrian is relatively still for a few seconds after halting, slight movements create specular reflections and shadows off the metallic background, inducing randomly placed change point vertices. As previously mentioned, these change point vertices and hulls will be filtered out after further analysis. When the pedestrian begins to move again, the path of the change point vertices in 3-d space-time follow a curved surface until the pedestrian reaches a constant velocity, at which time the change vertices are coplanar. Due to the direction of the pedestrian, from right to left, the coplanar change vertices are at an angle behind change point vertices associated with the pedestrian entering the scene.
Figure 4.19: *Walk-Forward* video sequence and change point hull segmentation. A single pedestrian walks straight through a scene at constant speed as seen in the example frames of figures 4.19a, 4.19b and 4.19c. At the beginning of the sequence, the DDHMM algorithm creates an initial states for the color of the wall background. When the pedestrian enters the scene, additional state-duration pairs are created to account for the pedestrians appearance and motion dynamics. The change point vertices are shown in figure 4.19d. Figures 4.19f and 4.19g show the change point hulls in 3D image space-time from a side and top view respectively where color denotes common hull membership and black is empty space. Figure 4.19e shows a straight on view of the change point vertices and hulls associated with the pedestrian over time.
Figure 4.20: *Stop-Forward* video sequence. Figures 4.20a through 4.20d show example frames taken from the Stop-Forward video. The scene contains a pedestrian walking from left to right, slowing and stopping for a few seconds at the center of the frame and walking out of the scene to the right. The resulting change point geometry is shown by the collection of change point vertices and change point hulls visualized in figures 4.20e and 4.20f where the color denotes hull membership.
Figure 4.21: Tracing the path of change point vertices created by a 1-d object that enters a scene traveling at a constant velocity from left to right, slows and stops, pausing for a time before accelerating and leaving the camera view at a constant velocity to the right. The path of change point vertices is shown in purple as a function of space and time. When the object enters the scene at relatively constant velocity, the vertices trace a line. As the object slows, the sequence of change point vertices curves upwards as less space is traveled per unit time. When the object stops, there are no additional change point vertices in the frame and the duration of the previous vertex is extended until the object begins to move again. Once the object begins to accelerate, new change point vertices trace a curve until a terminal constant velocity is reached and the vertices again follow a linear path.
Figure 4.22: Checker-Back video sequence. The scene contains specular metallic checkered background as seen in the example frames in figures 4.22a through 4.22d. The scene consists of a pedestrian which enters the camera’s view, initially traveling at a constant velocity from left to right. At approximately the frame center, the pedestrian slows to a halt and waits a few seconds prior to backing out of the scene in the direction he came from. Figures 4.22e through 4.22h visualize the change point vertices and change point hulls associated with the pedestrian in 3-d image space-time from different vantage points.
Figure 4.23: Idealized illustration of the change point geometry induced when imaging a 1-d that enters a scene with constant velocity, slows to a halt and pauses prior to accelerating to a constant velocity, exiting the scene from the direction it came. The sequence of change point vertices in 2-d space-time is shown in purple. As the object enters the scene with constant velocity, the change point vertices trace a line with positive slope. As the object slows, the change point vertices trace a curve as less space is traveled per unit time. While the object is at rest, no new change point vertices are created, leaving a gap in space-time. When the object accelerates to the left, the sequence of change point vertices trace a curve until a constant velocity is reached and the vertices trace a line. Due to the direction of the object, the latter linear set of vertices follows a line with negative slope compared to the line associated with the vertices created when the object entered the scene.
4.4 Change Point Hull Eigen-Shape

It is shown by the experiments in sections 4.2 and 4.3.2, change point hulls associated with object movement are composed of change point vertices that are locally coplanar points in 3-d space-time. Even in the case of the Fields Point turbine shown in figure 4.8, the change point vertices corresponding to the appearance discontinuities caused by the periodic rotation of the wind turbine are clustered into small change point hulls that are locally coplanar.

To describe the shape of a change point hull, the eigenvectors and eigenvalues of the scatter matrix of positions in 3-d space-time of the member change point vertices are computed. The collection of eigenvectors and eigenvalues are denoted,

$$E = \{ (E_i, \lambda_i) \mid \lambda_i \geq \lambda_j \ \forall \ i \leq j, \lambda \in \mathbb{R}, E_i \in \mathbb{R}^3, |E_i| = 1 \}$$ (4.32)

where $|E_i|$ is the magnitude of the $i^{th}$ eigenvector.

Given a collection of $N$ change point vertices locations, \( \{(x_i, y_i, z_i) \mid i \in \{1, \ldots, N\}\} \), the scatter matrix is a $4 \times 4$ square symmetric matrix defined as,

$$S = \begin{bmatrix}
\langle x^2 \rangle & \langle xy \rangle & \langle xz \rangle & \langle x \rangle \\
\langle xy \rangle & \langle y^2 \rangle & \langle yz \rangle & \langle y \rangle \\
\langle xz \rangle & \langle yz \rangle & \langle z^2 \rangle & \langle z \rangle \\
\langle x \rangle & \langle y \rangle & \langle z \rangle & N
\end{bmatrix}$$ (4.33)

where the notation \( \langle \cdot \rangle \) denotes the empirical expectation of the argument [81],

$$\langle w \rangle = \frac{1}{N} \sum_{i=1}^{N} w_i$$ (4.34)

The eigenvectors of the scatter matrix form an orthonormal basis in the directions of maximal variance of the point set and the eigenvalues are proportional to the variance of the point spread in the direction of the corresponding eigenvector [82]. Intuitively, if the points are coplanar, the smallest eigenvalue will be very close to zero and the normal of the best fitting plane will be the eigenvector paired with the smallest eigenvalue.
**Singular Value Decomposition** (SVD) is a standard technique for solving for the eigenvectors and eigenvalues of a matrix [82, 83, 81]. For a given change point hull, the member vertices are used to compute a scatter matrix and SVD is used to recover the eigenvectors and eigenvalues. The smallest eigenvalues serves as a measure of the average error between the change vertices and the best fitting plane. The eigenvector associated with the smallest eigenvalue contains information about the direction of object motion.

![Figure 4.24: 1-d example using the eigenvalues and eigenvectors as change point hull descriptors. The blue points correspond to change point vertices in 2-d space time that belong to the same change point hull. The red arrows are the eigenvectors of the change point vertex scatter matrix and their length is proportional to the magnitude of their corresponding eigenvalues. The eigenvector with the largest eigenvalue is parallel to the line that best fits the change point locations. The eigenvector with the smallest eigenvalue is perpendicular to the best fitting line.](image)

For example, again consider the change point vertices associated with a one dimensional object moving through space over time as visualized in figure 4.24. The green points are a set of change point vertices associated with an object moving from left two right along the spacial axis over time in the vertical axis. The eigenvectors and eigenvalues of the scatter matrix computed according to the change point vertices location in 2-d space time are recovered using SVD. The resulting eigenvectors are shown as the red arrows and their lengths are proportional to their associated eigenvalues. The
eigenvector with the largest eigenvalue will be parallel to the line that best fits the locations of the change point vertices. The line that best fits the points is the line that minimizes the sum of squared errors between the points and the line. The best fitting line is visualized by the dotted green line. The eigenvector with the smallest eigenvalue is perpendicular to the best fitting line. The components of the eigenvector with the smallest eigenvalue are equal to the coefficients of the best fitting line parameterized by the normal equation satisfying,

\[ \mathcal{E}_2 \cdot (x, t)^T = 0 \quad (4.35) \]

where \( \mathcal{E}_2 \in \mathbb{R}^2 \) in this example.

In the 3-d space-time, the eigenvector associated with the smallest eigenvalue will be normal to the best fitting plane embedded in 3-d space-time satisfying,

\[ \mathcal{E}_3 \cdot (x, y, z)^T = 0 \quad (4.36) \]

While all eigenvectors have unit magnitude, the eigenvector associated with the smallest eigenvalue, \( \mathcal{E}_3 \), will be referred to as the smallest eigenvector for brevity. Note there is an ambiguity regarding the orientation of the smallest eigenvector. In the 1-d example of figure 4.24, the smallest eigenvector could be reflected across the best fitting line resulting in an equivalent decomposition. Therefore, for change point vertices in 3-d space-time, the third component of the smallest eigenvector is negated if its value is greater than zero, establishing the convention that the smallest eigenvector should point toward the camera.
4.5 Coupling Duration Dependent Hidden Markov Models and Change Point Geometry for Video Surveillance

4.5.1 Computing the Direction of Moving Objects

As argued in section 4.4, for a set of coplanar points, the eigenvector with the smallest eigenvalue, $E_3$, of the scatter matrix will be normal to the plane that bests fits the points in terms of the sum of squared distances from the collection of points to the plane. Having adopted the convention that the smallest eigenvalue point toward the camera, if $E_3$ is the eigenvector corresponding to the smallest eigenvalue of the scatter matrix computed using the member vertices of a change point hull, the projection of $E_3$ into the video frame will point in the direction of lateral object motion.

For example, consider the 2-d space time plot of change point vertices corresponding the movement of 1-d objects in figures 4.25 and 4.26. Figure 4.25 shows the best fitting line normal, the normalized eigenvector with smallest eigenvalue, as green and blue lines superimposed on the corresponding change point vertices. The best-fitting line normals are projected on to the image plane in the corresponding colors just above the imaging line and reflect the direction of the object responsible for the change point vertex groups.

Note, the velocity of the object is inversly proportional to the magnitude of the projected eigenvector. Figure 4.26 shows a more extreme example in terms of object speed. The object responsible for the green change point vertices moves very slow while the object responsible for the blue set of vertices is moving quickly. The change point vertices associate with slowing moving 1-d objects trace a vertical line in 20d space time compared to objects moving at a more rapid pace which trace relatively horizontal lines. Therefore, the magnitude of the projection line normal into the image frame will be inversely proportional to the relative speed of the object.

Figure 4.27 shows the recovered motion vectors for the Crossing-Paths video sequence. The vectors corresponding to the change point detected on each pedestrian points in the correct direction.
Figure 4.25: Visualizing the direction of motion for one-dimensional objects. The green and blue dots correspond to change point vertices associated with the motion of two different 1-d objects. The green and blue arrows on the change points correspond to the eigenvector with smallest eigenvector computed using the scatter matrix of the locations of the change vertices in each group. The projection of the eigenvectors onto the camera plane is in the direction of object motion as visualized by the arrows just above the imaging line.

Figure 4.26: Visualizing the effects of object velocity on the magnitude of the projected eigenvectors. A 1-d object moving at a slow constant speed creates change point vertices that trace a vertical line compared to an object with greater lateral speed. The change point vertices associated with the first object, which is moving slowly, are visualized as green dots. The change point vertices associated with a second 1-d object moving quickly are visualized as blue dots. The projection of the normal of the best fitting lines with regards to both sets of change point vertices are shown right above the imaging line. The magnitude of the projected vectors is inversely proportional to object speed.

While the pedestrian moving from right to left casts a shadow on the wall creating some false change point vertices, the movement of the shadow from right to left is still captured by the projection of
the appropriate change point hull eigenvectors.

![Motion vectors recovered from the Crossing-Paths video.](image)

Figure 4.27: Motion vectors recovered from the Crossing-Paths video.

### 4.5.2 Detecting Change

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_w )</td>
<td>Threshold for smallest eigenvalue</td>
<td>0.3</td>
</tr>
<tr>
<td>( \tau_h )</td>
<td>Threshold for joint probability of normal direction and singular value</td>
<td>0.2</td>
</tr>
<tr>
<td>Cell Size</td>
<td>Image grid size</td>
<td>20 × 20</td>
</tr>
</tbody>
</table>

Table 4.6: Summary of proposed region-level change detection parameters.

This section describes a region-level change detection algorithm based on the distribution of the smallest eigenvectors and eigenvalues of change point hulls originally published in [77]. A background model is created by first dividing the image plane into a non-overlapping grid of cells. During a training phase, each change point hull is assigned to a grid cell according to the majority membership of its vertices. First, a threshold is applied to the smallest eigenvalue of the change point hull. If the smallest eigenvalue is greater than \( \tau_w \), it is assumed the hull does not fit a planar model and thus does not represent coherent motion and is discarded. For all remaining change point hulls, the
angle of the normal direction of the hull is computed with respect to the reference vector, \((1, 0)\), by projecting the smallest eigenvector of the data covariance matrix decomposition into the image plane. For each cell of the grid, a 2d histogram of normal directions and absolute values of the smallest eigenvalue is maintained as a background model.

During testing, the eigenvalue threshold \(\tau_w\) is again applied to each change hull, discarding hulls which do not fit a planar model. The remaining change hulls are assigned to grid cells and compared with the background histogram. Deviations from the background model are determined by thresholding the joint probability of a change hull’s normal direction and smallest singular value with respect to the background model by a constant \(\tau_h\). All vertices that comprise a hull deemed to be abnormal are labeled as change. In all experiments, the grid cell size was set to \(20 \times 20\) pixels, \(\tau_w = 0.3\) and \(\tau_h = 0.2\).

Figures 4.28 and 4.29 visualize the results of the proposed change detection system. Frames of the Walk-Forward video are shown again in the first row of figure 4.28. The subsequent rows show the change detection results of the proposed, SGMM-SOD and STLBP algorithms respectively. Change detections are visualized using binary masks, where a black pixel in a frame means the intensity at that time and location is considered to be normal scene dynamics and a white pixel represents a significant change was detected at the location. Considering the proposed system, figure 4.28, most of the person is detected as change. However, his pants are a very similar color to the hallway rendering it difficult to detect appearance discontinuities between the pants and the background resulting in false negatives on the pedestrians legs. The SGMM-SOD results are similar, exhibiting false positives on the pedestrians pants. Conversely, the STLBP algorithm contains many false positive. The wall is a homogeneous, texturless surface, rendering the gradient-like LBP descriptor very sensitive to compression artifacts, shadows and other noise in the imaging system. While STLBP is able to distinguish between the pedestrian’s pants from the wall due to the texture differences from the subtle shadows cast by the bending legs, the STLBP algorithm fails to accurately model the textureless wall.
Figure 4.29 shows the change detection results of the proposed region-level, SGMM-SOD, and STLBP algorithms applied to the Stop-Forward video. The first row, figure 4.29a again shows frames of the Stop-Forward video ordered from left to right. Each subsequent row shows the results of the noted change detection algorithm, visualized as a binary mask, for each frame in the corresponding column. A white pixel means a significant change is detected at the position whereas a black pixel indicates no change was detected at the location.

The change detection results of the proposed region-level algorithm captures are shown in figure 4.29b. The proposed algorithm detects most of the pedestrian, however, there are some false negatives on the pedestrian in the second frame owing to the similar color of the background to the pedestrian’s skin color. The results of the SGMM-SOD algorithm are shown in figure 4.29c. The SGMM-SOD algorithm results are slightly better than the proposed algorithm in this example but there are still some false negatives on the pedestrians neck and hands due to their similar appearance to the wall.

The STLBP fails to reliably detect changes in this scene. Just as in the Walk-Forward video, the LBP descriptor is ill-defined on the textureless regions of the wall. Additionally, the pedestrian in this experiment is wearing dark clothes with solid colors whereas in the Walk-Forward video, the pedestrian was wearing multi-colored shirt with stripes. The striped shirt creates an appearance texture and a corresponding LBP code that is unique compared to codes generated on the homogeneous wall. Because of the solid dark colors of the pedestrian’s clothes in the Stop-Forward video, the only significant appearance discontinuities in the frames of video are at the boarder between the pedestrian and the wall and the interior of the pedestrian is relatively homogeneous and textureless. Therefore, the STLBP algorithm struggles to make a distinction between the pedestrian and the wall, resulting in a large amount of false negatives. Simultaneously, STLBP algorithm still struggles to model the homogeneous background and continues to make random false positive detections as in the Walk-Forward scene.
Figure 4.28: Walk-Forward region-level change detection results. The first row of images, figure 4.28a show example frames of video from the Walk-Forward sequence in order from left to right. Figures 4.28b, 4.28c and 4.28d visualize the change detection results of the proposed region-level, SGMM-SOD and STLBP algorithms respectively for the frame in the corresponding column. A white pixel represents a significant change and a black pixel does not.
Figure 4.29: Change detection results for region level algorithms using the Stop-Forward video. The first row of images, figure 4.29a show frames from the Stop-Forward video in order from left to right. Figures 4.29b, 4.29c and 4.29d visualize the change detection results of the proposed region-level, SGMM-SOD and STLBP algorithms respectively. Each change detection mask corresponds to the output for the frame in the same column. A white pixel indicates a significant change was detected in the corresponding location whereas a black pixel indicates no change was detected in that area.
Chapter 5

Conclusions and Future Work

This thesis has developed two novel algorithms for automatically detecting meaningful changes in surveillance video. First, intensity sequences at each pixel location in a video frame are modeled using independent duration dependent hidden Markov models at each pixel. A novel, computationally efficient learning algorithm was proposed that was shown to more accurately model intensity sequences from real-world video surveillance footage than analogous state of the art methods based on Gaussian mixtures and hidden Markov models. At each pixel location, a codebook, composed of state-duration assignments made by the DDHMM learning algorithm is used in a supervised change detection application. Compared with other state of the art change detection methods, the proposed algorithm is the only one which can learn that periodic motion may be apart of normal scene dynamics and still accurately localize novel phenomena.

A region-level algorithm for scene understanding and change detection was also developed based on the original duration dependent hidden Markov pixel intensity representation. In this framework, state-duration assignments made by independent DDHMMs at each pixel are used to implicitly segment pixel intensity sequences into regions of similar appearance. The location of the start of homogeneous regions that differ significantly from the previous and next homogeneous sequences in time, may be thought of as point in 3-d image space-time. The coordinates of the start of
homogeneous subsequences augmented with a state-duration assignment are referred to as a change point vertex.

Change point vertices are clustered using a graph-segmentation algorithm to group change point vertices that represent subsequences with similar appearance and duration statistics that are close in space-time. Groups of change point vertices are called change point hulls. The set of change point hulls represent groups of intensity subsequences that are self-similar with respect to member of the same change point hull, but are significantly different compared to subsequences in neighboring change point hulls. The result is a segmentation that may be thought of as a collection of "super-voxels" of appearance discontinuities in 3-d space-time. It was shown experimentally that point clouds of change point vertices associated with moving objects are locally coplanar while vertices associated with random image noise, reflections or other stochastic imaging effects are scattered and not coplanar in 3-d space-time.

For each change point hull and its member vertices, the smallest eigenvalue of the scatter matrix computed using the positions of the change point vertices is used to determine if the change point vertices of each hull are coplanar. If a change point hull’s vertices are consider coplanar, the direction and relative velocity of object motion may be inferred. Additionally, a classifier based on the joint probability distribution of the best fitting plane normal of each change point hull and its smallest eigenvalue is used to detect change.

Within the proposed framework, there are many avenues to peruse further research. The per-pixel temporal and region-level algorithms are currently independent. A natural extension would be to incorporate a feedback mechanism to allow for the region-level analysis to help drive per-pixel temporal DDHMM state assignments.

Additionally, the change point hull segmentation step could incorporate a shape constraint to cluster similar vertices that are also coplanar. Within the context of the graph-segmentation algorithm proposed by Felzenszwalb and Huttenlocker [23], the regularizing term of the minimal internal difference, $\tau(C)$, could be use to incorporate a shape constraint directly into the inter-component
similarity measure. However, developing a shape constraint that prefers coplanar points while leaving the computational complexity of the original algorithm unchanged may prove difficult.
Appendix A

Expectation Maximization

The *Expectation Maximization* (EM) algorithm, is an iterative local parameter search algorithm that is guaranteed to increase the likelihood function with each step and converge to a local optimum [46, 47, 45, 84]. While the Expectation Maximization algorithm is a general estimation framework for latent variable models, the focus of the proceeding review is direct derivation of the EM algorithm for the case of Gaussian Mixture and Hidden Markov models.

A.1 Expectation Maximization for GMMs

For a given dataset, \( \mathbb{Y} = \{ y_1, \ldots, y_T \} \), the goal is to estimate a set of parameters for the GMM, \( \Theta = \{ \{ \hat{\mu}_k, \hat{\sigma}_k, \hat{\omega}_k \} \mid k \in \{1, \ldots, K\} \} \) that maximizes the likelihood of the data with respect to the model. Recall the likelihood function for a univariate Gaussian mixture model with \( K \) states is:

\[
L(y_1, \ldots, y_T \mid \Theta) = \prod_{t=1}^{T} \sum_{k=0}^{K} \omega_k \mathcal{N}(y_t \mid \mu_k, \sigma_k)
\]  

(A.1)

where \( \mathcal{N}(y_t \mid \theta_k) \) is shorthand for the Gaussian or Normal distribution of equation 2.3. Because the logarithm is a monotonic function, equation A.1 can be maximized with respect to the model parameters by taking the derivative of the log-likelihood with respect to each parameter, setting
to zero and solving. The logarithm is used for algebraic simplicity, transforming products into summations making the log-likelihood function easier to differentiate.

\[
\frac{\partial}{\partial \mu_k} \ln L (y_1, \ldots, y_T \mid \Theta) = \frac{\partial}{\partial \mu_k} \sum_{t=1}^{T} \ln \left( \sum_{k=1}^{K} \omega_k \mathcal{N}(y_t \mid \mu_k, \sigma_k) \right) = \sum_{t=1}^{T} \frac{\omega_k \mathcal{N}(y_t \mid \mu_k, \sigma_k)}{\sum_{j=1}^{K} \omega_j \mathcal{N}(y_t \mid \mu_j, \sigma_j)} \sigma_k (y_t - \mu_k) \tag{A.2}
\]

In the manner of [45], the remaining equations are simplified by defining:

\[
\gamma_{tk} = \frac{\omega_k \mathcal{N}(y_t \mid \mu_k, \sigma_k)}{\sum_{j=1}^{K} \omega_j \mathcal{N}(y_t \mid \mu_k, \sigma_k)} \tag{A.3}
\]

\[
N_k = \sum_{t=1}^{T} \gamma_{tk} \tag{A.4}
\]

Where \(\gamma_{tk}\) may be thought of as the probability of associating state \(k\) with the \(t^{th}\) observation and \(N_k\) is a “soft” count of the number of observations associated with state \(k\). Setting equation (A.2) to zero and solving gives the maximum likelihood solution for the mean parameter of each component:

\[
\sum_{t=1}^{T} \gamma_{tk} \sigma_k (y_t - \mu_k) = 0 \\
\sigma_k \sum_{t=1}^{T} \gamma_{tk} y_t - \mu_k \sigma_k \sum_{t=1}^{T} \gamma_{tk} = 0 \\
\mu_k = \frac{1}{N_k} \sum_{t=1}^{T} \gamma_{tk} y_t \tag{A.5}
\]

Likewise, the partial derivative of the log-likelihood function with respect to the component variance, equating to zero, and solving yields:

\[
\sigma_k^2 = \frac{1}{N_k} \sum_{t=1}^{T} (y_t - \mu_k)^2 \tag{A.6}
\]

Optimizing the GMM log-likelihood function with respect to the component mixture weights \(\{\omega_1, \ldots, \omega_K\}\) requires special consideration of the normalization constraint:

\[
\sum_{k=1}^{K} \omega_k = 1 \tag{A.7}
\]
Therefore, the unconstrained optimization must be augmented with a Lagrange multiplier to satisfy constraint A.7. Defining the Lagrangian functional:

$$\Lambda (L, \lambda) = \sum_{t=1}^{T} \ln \left[ \sum_{j=1}^{K} \omega_j N \left( y_t \mid \mu_j, \sigma_j \right) \right] + \lambda \left( \sum_{j=1}^{K} \omega_j - 1 \right)$$

(A.8)

The partial derivative of the lagrangian with respect to $$\omega_k$$ is:

$$\frac{\partial \Lambda (L, \lambda)}{\partial \omega_k} = \sum_{t=1}^{T} \frac{N \left( y_t \mid \mu_k, \sigma_k \right)}{\sum_{j=1}^{K} \omega_j N \left( y_t \mid \mu_j, \sigma_j \right)} + \lambda$$

(A.9)

First, equation A.9 is equated to zero and solved for the lagrange multiplier $$\lambda$$.

$$\sum_{t=1}^{T} \frac{N \left( y_t \mid \mu_k, \sigma_k \right)}{\sum_{j=1}^{K} \omega_j N \left( y_t \mid \mu_j, \sigma_j \right)} + \lambda = 0$$

(A.10)

Multiplying both sides by $$\sum_{k=1}^{K} \omega_k$$:

$$\sum_{t=1}^{T} \sum_{k=1}^{K} \frac{\omega_k N \left( y_t \mid \mu_k, \sigma_k \right)}{\sum_{j=1}^{K} \omega_j N \left( y_t \mid \mu_j, \sigma_j \right)} + \lambda \sum_{k=1}^{K} \omega_k = 0$$

$$\lambda = -T$$

(A.11)

Substituting $$\lambda = -T$$ into equation A.9 and solving for $$\omega_k$$ gives the optimal solution for each component weight with respect to the GMM log-likelihood.

$$\sum_{t=1}^{T} \frac{N \left( y_t \mid \mu_k, \sigma_k \right)}{\sum_{j=1}^{K} N \left( y_t \mid \mu_j, \sigma_j \right)} - T = 0$$

$$\frac{1}{\omega_k} \sum_{t=1}^{T} \gamma_{tk} = T$$

$$\omega_k = \frac{N_k}{T}$$

(A.12)

In summary, the log-likelihood is optimized by the set of coupled equations:

$$\mu_k = \frac{1}{N_k} \sum_{t=1}^{T} \gamma_{tk} y_t$$

(A.13)

$$\sigma^2_k = \frac{1}{N_k} \sum_{t=1}^{T} \gamma_{tk} \left( y_t - \mu_k \right)^2$$

(A.14)

$$\omega_k = \frac{N_k}{T}$$

(A.15)
Equations A.13, A.14 and A.15 cannot be solved in closed form, therefore the EM algorithm iterates between two steps aptly called the $E$ and $M$ steps, incrementally increasing the GMM likelihood function with each step. During the $E$ step, the terms $\gamma_{tk}$ are estimated for a fixed setting of parameter estimates. During the $M$, the parameter estimates are updated according to equations A.13, A.14 and A.15 using the recomputed values of $\gamma_{tk}$ from the $E$ step. The process repeats until a maximum number of iterations is reached, or the change of the GMM likelihood between iterations of the EM algorithm is less than a threshold $\epsilon$. Pseudocode for the EM algorithm for the Gaussian mixture model is given in listing 10.

**Algorithm 10 Expectation Maximization Algorithm for GMMs**

```
Require: $Y = \{y_1, \ldots, y_T\}, K, \epsilon, \text{nitr}$
1: Initialize parameter estimates $\hat{\Theta} = \{\{\hat{\mu}_k, \hat{\sigma}_k, \hat{\omega}_k\} | k \in \{1, \ldots K\}\}$
2: $i \leftarrow 1$
3: converged $\leftarrow$ False
4: while $i < \text{nitr}$ and not converged do
5:     for all $t \in \{1, \ldots, T\}$ do $\triangleright E$ step
6:         for all $k \in \{1, \ldots, K\}$ do
7:             $\gamma_{tk} \leftarrow \frac{\omega_k N(y_t | \mu_k, \sigma_k)}{\sum_{j=1}^{K} \omega_j N(y_t | \mu_j, \sigma_j)}$
8:         end for
9:     end for
10:    for all $k \in \{1, \ldots, K\}$ do $\triangleright M$ step
11:        $\hat{\mu}_{k}^{\text{new}} \leftarrow \frac{1}{N_k} \sum_{t=1}^{T} \gamma_{tk} y_t$
12:        $\hat{\sigma}_{k}^{\text{new}} \leftarrow \sqrt{\frac{1}{N_k} \sum_{t=1}^{T} \gamma_{tk} (y_t - \mu_k)^2}$
13:        $\hat{\omega}_{k}^{\text{new}} \leftarrow \frac{N_k}{T}$
14:    end for
15:    $\hat{\Theta}^{\text{new}} \leftarrow \{\{\hat{\mu}_k, \hat{\sigma}_k, \hat{\omega}_k\} | k \in \{1, \ldots, K\}\}$
16:    if $L \left( Y \mid \hat{\Theta}^{\text{new}} \right) - L \left( Y \mid \hat{\Theta} \right) \leq \epsilon$ then
17:     converged $\leftarrow$ True
18: end if
19: $\hat{\Theta} \leftarrow \hat{\Theta}^{\text{new}}$
20: $i \leftarrow i + 1$
21: end while
22: return $\hat{\Theta}$
```
A.2 Expectation Maximization for HMMs

This section presents an informal review of the fundamental concepts driving the EM algorithm for estimating the parameters of a HMM, more detailed discussions may be found in [56, 44, 45]. Direct optimization of the HMM log-likelihood is more difficult than the GMM due to the state transition distribution which prohibits factoring the likelihood over the length of the observation sequence. However, the EM algorithm for HMM uses a dynamic programming algorithm as a subroutine to compute the posterior latent state and transition probabilities with a complexity that is linear in the number of observations. The posterior probabilities then are used to update the HMM parameter estimates. This process is iterated and can be shown to increase the likelihood of the HMM at each step. Recall the parameters of a HMM with univariate Gaussian emission densities consist of a vector of normalized initial state probabilities \( \pi = \{ \pi_k \mid k \in \{1, \ldots, K\} \} : \sum_{k} \pi_k = 1 \), a row normalized matrix of transition probabilities \( A = \{ A_{i,j} \mid i, j \in \{1, \ldots, K\} \} \), \( A_{i,j} \in [0, 1] \), \( \sum_{j} A_{ij} = 1 \} \) and the state Gaussian emission parameters \( \{ \{ \mu_k, \sigma_k \} \mid k \in \{1, \ldots, K\} \} \). It may be shown that the maximum likelihood update equations for the parameters of a univariate Gaussian HMM are [56, 45] are given by:

\[
\begin{align*}
\pi_k &= \frac{\gamma_{tk}}{\sum_{j=1}^{K} \gamma_{1j}} \quad \text{(A.16)} \\
A_{jk} &= \frac{\sum_{t=1}^{T} \xi_{tjk}}{\sum_{t=1}^{T} \sum_{i=2}^{K} \xi_{tij}} \quad \text{(A.17)} \\
\mu_k &= \frac{\sum_{t=1}^{T} \gamma_{tk} y_t}{\sum_{t=1}^{T} \gamma_{tk}} \quad \text{(A.18)} \\
\sigma_k &= \frac{\sum_{t=1}^{T} \gamma_{tk} (y_t - \mu_k)^2}{\sum_{t=1}^{T} \gamma_{tk}} \quad \text{(A.19)}
\end{align*}
\]

Where \( \gamma_{tk} \) is the posterior probability of the state assignment at time \( t \), \( p(s_t \mid Y, \Theta) \), and \( \xi_{tjk} \) is the joint posterior probability of observing a transition from state \( j \) to \( k \) at time \( t \), \( p(s_{t-1}, s_t \mid Y, \Theta) \). The E step of the EM algorithm consists of computing \( \gamma_{tk} \) and \( \xi_{tik} \) while the M step recomputes the parameter estimates according to equations A.16, A.17, A.18 and A.19 using posterior probabilities from the E step. The difficulty arises in computing \( \gamma_{tk} \) and \( \xi_{tik} \) which by brute force would require
\( \mathcal{O}(K^T) \) operations but may be computed in \( \mathcal{O}(KT) \) time using the following dynamic program [56, 45].

Using Bayes rule, the posterior probability of a state at a given time is given by:

\[
\gamma_{tk} = p(S_t = s_k \mid \mathcal{Y}) = \frac{p(\mathcal{Y} \mid S_t = s_k)p(S_t = s_k)}{p(\mathcal{Y})}
\]  

where the notation \( p(S_t = s_k \mid \mathcal{Y}) \) denotes the probability the random variable state indicator at time \( t \), \( S_t \) takes on the state \( s_k \) given the observation at time \( t \), \( y_t \).

Either by inspection of the graphical model associated with the HMM or by algebraic manipulation, it may be shown that due to the temporal independence assumptions imposed by the HMM, the posterior probability of the state assignments factors according to [43, 44, 45]:

\[
p(S_t \mid \mathcal{Y}) = \frac{p(y_1, \ldots, y_t, S_t)p(y_{t+1}, \ldots, y_T \mid S_t)}{p(\mathcal{Y})} = \frac{\alpha_{(t,k)}\beta_{(t,k)}}{p(\mathcal{Y})} \tag{A.21}
\]

where \( \alpha_{(t,k)} \) and \( \beta_{(t,k)} \), known as the forward and backward variables respectively, are defined as:

\[
\alpha_{(t,k)} = p(y_1, \ldots, y_t, S_t = s_k) \tag{A.22}
\]

\[
\beta_{(t,k)} = p(y_{t+1}, \ldots, y_T \mid S_t = s_k) \tag{A.23}
\]

Note, for every value of \( t \), the forward and backward variables are each a list of \( K \) numbers, one for each value of \( S_t \). Both the forward and backward variables may be computed recursively [56, 45] according to

\[
\alpha_{(t,k)} = p(y_t \mid S_t = s_k) \sum_{j=1}^{K} \alpha_{(t-1,j)}p(S_t = s_k \mid S_{t-1} = s_j) \tag{A.24}
\]

\[
\beta_{(t,k)} = \sum_{j=1}^{K} \beta_{(t+1,j)}p(y_{t+1} \mid s_{t+1} = j)p(s_{t+1} = j \mid s_t = k) \tag{A.25}
\]

where the base cases for the forward and backward recursions are

\[
\alpha_{(1,k)} = p(y_1, S_1) = \prod_{k=1}^{K} \pi_k p(y_1 \mid s_k) \tag{A.26}
\]

\[
\beta_{(t,k)} = 1 \quad \forall \ k \tag{A.27}
\]
Examining the forward and backward recurrence relations of equations A.24 and A.25, each of the $K$ values of $\alpha_{(t,s_t)}$ and $\beta_{(t,s_t)}$ at any time, require a summation over $K$ terms, requiring $O(K^2)$ operations per time step.

In terms of the forward-backward program, following the cancellation of the data likelihood term present in the numerator and denominator, the emission parameters may be updated as:

\[
\mu_k = \frac{\sum_{t=1}^{T} \alpha_{(t,k)} \beta_{(t,k)} y_t}{\sum_{t=1}^{T} \alpha_{(t,k)} \beta_{(t,k)}} \quad \text{(A.28)}
\]

\[
\sigma_k = \frac{\sum_{t=1}^{T} \alpha_{(t,k)} \beta_{(t,k)} (y_t - \mu_k)^2}{\sum_{t=1}^{T} \alpha_{(t,k)} \beta_{(t,k)}} \quad \text{(A.29)}
\]

The data likelihood term may likewise be evaluated in terms of the forward-backward variables either at an arbitrary time $t$ or in terms of the forward variables run through the length of the chain:

\[
p(Y) = \sum_{k=1}^{K} \alpha_{(t,k)} \beta_{(t,k)}
\]

\[
= \sum_{k=1}^{K} \alpha_{(T,k)} \quad \text{(A.30)}
\]

and the joint posterior transition probabilities may be expressed in terms of the dynamic program [56, 45]:

\[
\xi_{t,i,j} = \frac{\alpha_{(t-1,i)} p(y_t \mid S_t = s_j) p(S_t = s_j \mid S_{t-1} = s_i) \beta_{(t,j)}}{p(Y)}
\]

\[
= \frac{\alpha_{(t-1,i)} p(y_t \mid S_t = s_j) p(S_t = s_j \mid S_{t-1} = s_i) \beta_{(t,j)}}{\sum_{k=1}^{K} \alpha_{(t,k)} \beta_{(t,k)}} \quad \text{(A.31)}
\]

A final, practical issue is must be considered before implementing the forward-backward algorithm for EM estimation. The forward and backward variables are functions of terms which involve $O(T)$ products of probabilities, numbers between zero and one. Therefore, it is very likely that for any observation sequence of non-trivial length, there will be problems with numerical underflow. The solution is to scale the forward and backward variables at each step by a factor which leaves the final computations invariant to the scaling but keeps the variable values in a reasonable range.
The normalized forward $\hat{\alpha}_{(t,k)}$ variables are defined as [56, 45]

$$\hat{\alpha}_{(t,k)} = \frac{p(y_1, \ldots, y_t \mid s_k)}{p(y_1, \ldots, y_t)}$$

$$\hat{\alpha}_{(t,k)} = \frac{\alpha_{(t,k)}}{p(y_1, \ldots, y_t)}$$ (A.32)

The scaled forward variables at each step are a normalized conditional probability distribution over $S_t$, rendering the scaled version of the forward variables within a reasonable range. Defining the scale factor variables as $c_t = p(y_t \mid y_1, \ldots, y_{t-1})$, a single number which normalizes the original forward variables at every time step, the scaled backward variables can be defined in terms of values of $c_t$ computed during the forward program

$$\hat{\beta}_{(t,k)} = \frac{\beta_{(t,k)}}{\prod_{j=t+1}^{T} c_j}$$ (A.33)

It may be shown that $\hat{\beta}_{(t,k)}$ is the ratio of two conditional probability distributions taking values in the range of modern machine precision [56, 45].

Pseudo code outlining the forward and backward programs are given in listings 11 and 12. The forward and backward algorithms are subroutines called during the E step of the EM algorithm for computing the posterior state and transition probabilities for a fixed parameter estimate. During the M step, the posterior probabilities from the E step are used to recompute new parameter estimates. This process repeats until the difference of the HMM likelihood between successive iterations of both the E and M steps falls below a threshold or a maximum number of iterations is reached. Pseudo code for the EM algorithm is given in listing 13.
Algorithm 11  Forward Program for HMMs

1: function FORWARD_HMM($\mathcal{Y}, \hat{\Theta}$)
2:     $F \leftarrow \emptyset$ \hfill $\triangleright$ Collection of $T \times K$ forward variables
3:     $c \leftarrow \emptyset$ \hfill $\triangleright$ Collection of $T$ scale factors
4:     for all $k \in \{1, \ldots, K\}$ do
5:         $\hat{\alpha}_{(1,k)} \leftarrow \prod_{k=1}^{K} \omega_k p(y_1 | \hat{\theta}_k)$
6:         $c_1 \leftarrow \sum_{k=1}^{K} \hat{\alpha}_{(1,k)}$\hfill $\hat{\alpha}_{(1,k)} \leftarrow \frac{\hat{\alpha}_{(1,k)}}{c_1}$
7:     end for
8:     $F \leftarrow \{\{\hat{\alpha}_{(1,k)} | k \in \{1, \ldots, K\}\} \cup F\}$
9:     $c \leftarrow \{c_1 \cup c\}$
10:    for all $t \in \{2, \ldots, T\}$ do
11:        for all $k \in \{1, \ldots, K\}$ do
12:            $\hat{\alpha}_{(t,k)} \leftarrow p(y_t | \hat{\theta}_k) \sum_{j=1}^{K} \hat{\alpha}_{(t-1,k)} \hat{A}_{jk}$
13:        end for
14:        $c_{\text{prod}} = c_T$
15:        for all $k \in \{1, \ldots, K\}$ do
16:            $\hat{\beta}_{(t,k)} \leftarrow \frac{1}{c_T} \sum_{k=1}^{K} \hat{\beta}_{(t+1,k)} p(y_{t+1} | S_{t+1} = s_k) \hat{A}_{jk}$
17:        end for
18:        $B \leftarrow \{\{\hat{\beta}_{(t,k)} | k \in \{1, \ldots, K\}\} \cup B\}$
19:    end for
20:    return $\{F, c\}$
21: end function

Algorithm 12  Backward Program for HMMs

1: function BACKWARD_HMM($\mathcal{Y}, \hat{\Theta}, c$)
2:     $B \leftarrow \emptyset$ \hfill $\triangleright$ Collection of $T \times K$ backwards variables
3:     for all $k \in \{1, \ldots, K\}$ do
4:         $\beta_{(T,k)} = \frac{1}{c_T}$
5:         $B \leftarrow \{\beta_{(T,k)} \cup B\}$
6:     end for
7:     $c_{\text{prod}} = c_T$
8:     for all $t \in \{T-1, \ldots, 1\}$ do
9:         $c_{\text{prod}} \leftarrow c_{t} c_{\text{prod}}$
10:        for all $k \in \{1, \ldots, K\}$ do
11:            $\hat{\beta}_{(t,k)} \leftarrow \frac{1}{c_{\text{prod}}} \sum_{k=1}^{K} \hat{\beta}_{(t+1,k)} p(y_{t+1} | S_{t+1} = s_k) \hat{A}_{jk}$
12:        end for
13:        $B \leftarrow \{\{\hat{\beta}_{(t,k)} | k \in \{1, \ldots, K\}\} \cup B\}$
14:    end for
15: return $B$
16: end function
Algorithm 13 EM Algorithm for HMMs

1: function EM_HMM(Y, nitr, $\epsilon$)
2: Initialize parameter estimates $\hat{\Theta}$
3: converged $\leftarrow$ False
4: $i \leftarrow 1$
5: while $i \leq nitr$ and not converged do
6: 
7: E-step: Compute posterior probabilities for current parameter estimates
8: Compute Forward Variables and scale factors $F, c \leftarrow \text{FORWARD_HMM}(Y, \hat{\Theta})$
9: Compute Backward Variables $B \leftarrow \text{BACKWARD_HMM}(Y, \hat{\Theta}, c)$
10: for all $t \in \{1, \ldots, T\}$ do
11: for all $k \in \{1, \ldots, K\}$ do
12: $\hat{\gamma}(t,k) \leftarrow C_t \hat{\alpha}(t-1,k) p\left(y_t \mid \hat{\theta}_k\right) \hat{A}_{ij} \hat{\beta}(t,k)$
13: end for
14: end for
15: 
16: M-Step: Update parameter estimates using current posteriors
17: for all $k \in \{1, \ldots, K\}$ do
18: $\hat{\omega}_k \leftarrow \sum_{t=1}^{T} \frac{\hat{\gamma}(t,k)}{\sum_{j=1}^{K} \hat{\gamma}(t,j)}$
19: $\hat{\mu}_k \leftarrow \sum_{t=1}^{T} \frac{\hat{\gamma}(t,k) y_t}{\sum_{j=1}^{K} \hat{\gamma}(t,j)}$
20: $\hat{\sigma}_k \leftarrow \frac{\sum_{t=1}^{T} \hat{\gamma}(t,k) (y_t - \hat{\mu}_k)^2}{\sum_{t=1}^{T} \hat{\gamma}(t,k)}$
21: $\hat{A}_{(i,j)} = \frac{\sum_{t=2}^{T} \hat{\xi}(t,j,i)}{\sum_{t=1}^{T} \sum_{j=1}^{K} \hat{\xi}(t,i,j)}$
22: end for
23: $\hat{\Theta}_{\text{new}} \leftarrow \{(\hat{\omega}_k, \hat{\mu}_k, \hat{\sigma}_k) \mid k \in \{1, \ldots, K\}\} \cup \{\hat{A}_{(i,j)} \mid i,j \in \{1, \ldots, K\}\}$
24: if $p\left(Y \mid \hat{\Theta}_{\text{new}}\right) - p\left(Y \mid \hat{\Theta}\right) \leq \epsilon$ then
25: converged $\leftarrow$ True
26: end if
27: $i \leftarrow i + 1$
28: $\hat{\Theta} \leftarrow \hat{\Theta}_{\text{new}}$
29: end while
30: return $\hat{\Theta}$
31: end function
Bibliography


2005.


