


Winter 2011

Data mining based learning algorithms for semi-supervised object identification and tracking

Michael P. Dessauer

Follow this and additional works at: <https://digitalcommons.latech.edu/dissertations>

 Part of the [Computer Sciences Commons](#), and the [Other Computer Engineering Commons](#)

**DATA MINING BASED LEARNING ALGORITHMS FOR
SEMI-SUPERVISED OBJECT IDENTIFICATION AND
TRACKING**

by

Michael P. Dessauer, B.S., M.S.

A Dissertation Presented in the Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

February 2011

UMI Number: 3451089

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

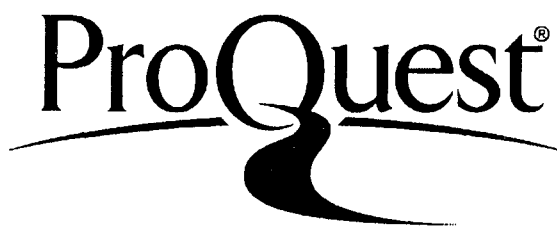
In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 3451089

Copyright 2011 by ProQuest LLC.

All rights reserved. This edition of the work is protected against unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

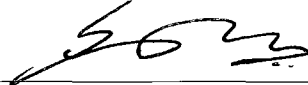
November 3, 2010

Date

We hereby recommend that the thesis prepared under our supervision
by Michael P. Dessauer

entitled Data Mining based Learning Algorithms for Semi-supervised Object Identification and Tracking

be accepted in partial fulfillment of the requirements for the Degree of
Doctor of Philosophy in Computational Analysis and Modeling



Supervisor of Thesis Research

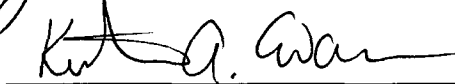
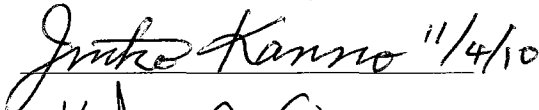
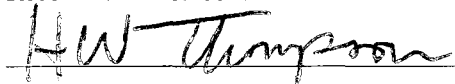


Head of Department

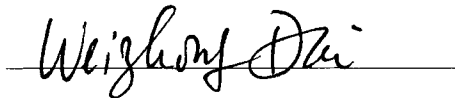
Computational Analysis and Modeling

Department

Recommendation concurred in:



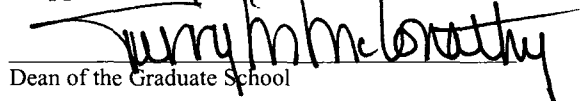
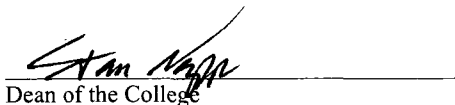
Advisory Committee



Approved:


Director of Graduate Studies

Approved:


Dean of the Graduate School
Dean of the College

ABSTRACT

Sensor exploitation (SE) is the crucial step in surveillance applications such as airport security and search and rescue operations. It allows localization and identification of movement in urban settings and can significantly boost knowledge gathering, interpretation and action. Data mining techniques offer the promise of precise and accurate knowledge acquisition techniques in high-dimensional data domains (and diminishing the “curse of dimensionality” prevalent in such datasets), coupled by algorithmic design in feature extraction, discriminative ranking, feature fusion and supervised learning (classification). Consequently, data mining techniques and algorithms can be used to refine and process captured data and to detect, recognize, classify, and track objects with predictable high degrees of specificity and sensitivity.

Automatic object detection and tracking algorithms face several obstacles, such as large and incomplete datasets, ill-defined regions of interest (ROIs), variable scalability, lack of compactness, angular regions, partial occlusions, environmental variables, and unknown potential object classes, which work against their ability to achieve accurate real-time results. Methods must produce fast and accurate results by streamlining image processing, data compression and reduction, feature extraction, classification, and tracking algorithms. Data mining techniques can sufficiently address these challenges by implementing efficient and accurate dimensionality reduction with feature extraction to

refine incomplete (ill-partitioning) data-space and addressing challenges related to object classification, intra-class variability, and inter-class dependencies.

A series of methods have been developed to combat many of the challenges for the purpose of creating a sensor exploitation and tracking framework for real time image sensor inputs. The framework has been broken down into a series of sub-routines, which work in both series and parallel to accomplish tasks such as image pre-processing, data reduction, segmentation, object detection, tracking, and classification. These methods can be implemented either independently or together to form a synergistic solution to object detection and tracking.

The main contributions to the SE field include novel feature extraction methods for highly discriminative object detection, classification, and tracking. Also, a new supervised classification scheme is presented for detecting objects in urban environments, which incorporates both novel features and non-maximal suppression to reduce false alarms, which can be abundant in cluttered environments such as cities. Lastly, a performance evaluation of Graphical Processing Unit (GPU) implementations of the subtask algorithms is presented, which provides insight into speed-up gains throughout the SE framework to improve design for real time applications.

The overall framework provides a comprehensive SE system, which can be tailored for integration into a layered sensing scheme to provide the war fighter with automated assistance and support. As more sensor technology and integration continues to advance, this SE framework can provide faster and more accurate decision support for both intelligence and civilian applications.

APPROVAL FOR SCHOLARLY DISSEMINATION

The author grants to the Prescott Memorial Library of Louisiana Tech University the right to reproduce, by appropriate methods, upon request, any or all portions of this Dissertation. It is understood that "proper request" consists of the agreement, on the part of the requesting party, that said reproduction is for his personal use and that subsequent reproduction will not occur without written approval of the author of this Dissertation. Further, any portions of the Dissertation used in books, papers, and other works must be appropriately referenced to this Dissertation.

Finally, the author of this Dissertation reserves the right to publish freely, in the literature, at any time, any or all portions of this Dissertation.



Author Michael P. Dessauer

Date 11/03/2010

TABLE OF CONTENTS

ABSTRACT.....	ii
LIST OF TABLES.....	ix
LIST OF FIGURES	x
ACKNOWLEDGEMENTS.....	xiv
CHAPTER 1 - INTRODUCTION.....	1
1.1 What is Object Recognition?	1
1.2 Data Mining and the Knowledge Discovery Process	4
1.3 Data Mining Based SE Method	6
1.4 Dissertation Outline	8
CHAPTER 2 - SENSOR EXPLOITATION FRAMEWORK.....	10
2.1 SE Framework Introduction.....	10
2.2 Image Preprocessing	11
2.3 Object Detection	14
2.4 Feature Extraction and Classification	16
2.5 Object Tracking	19
2.6 SE Framework Description Conclusion.....	21
CHAPTER 3 - IMAGE PREPROCESSING	22
3.1 Image Preprocessing Introduction	22
3.2 The Wiener Filter	24

3.3 Histogram Specification.....	25
3.4 Wavelet Decomposition.....	27
3.5 Gaussian Smoothing	29
CHAPTER 4 - OBJECT DETECTION.....	32
4.1 Object Detection Introduction.....	32
4.2 Motion Detection Using Optical Flow.....	33
4.2.1 Lucas and Kanade Method.....	33
4.2.2 Pyramid Implementation of Lucas and Kanade	35
4.2.3 Phase Based Optical Flow	36
4.2.4 Optical Flow Evaluation Conclusions	41
4.3 Object Detection Using Hierarchical Graph Segmentation	41
4.3.1 Graph Initialization	43
4.3.2 Node Energy Function	43
4.3.3 Graph Coarsening.....	44
4.3.4 Higher-level Node Statistics.....	46
4.3.5 Object Candidate Selection	48
4.3.6 Segmentation Evaluation and Results.....	49
4.3.7 Conclusion.....	51
4.4 Data-Mining Based Moving Object Detection	51
4.4.1 Motion Detection.....	52
4.4.2 Flow Gradient Intersect Features	53
4.4.3 Edge Intersect Features	54
4.4.4 Hierarchical Graph Segmentation	55

4.4.5	Object Candidate Selection	57
4.4.6	Supervised Classification Object Detection	58
4.4.7	Multi Frame Object Matching	59
4.4.8	Object Non Maximal Suppression	60
4.4.9	Object Detection Testing Data	61
4.4.10	Object Detection Evaluation	61
4.4.11	Conclusion.....	63
4.5	Object Detection Conclusion	64
CHAPTER 5 - FEATURE EXTRACTION AND CLASSIFICATION		65
5.1	Feature Extraction and Classification Introduction	65
5.2	Dense and Reduced Speed-up Robust Features (DRSURF).....	66
5.2.1	DRSURF Interest Point Detection	66
5.2.2	DRSURF Orientation Assignment	68
5.2.3	DRSURF Feature Descriptor Calculation	68
5.3	Invariant Moment Descriptors	70
5.4	Edge Histograms.....	72
5.5	Texture Statistics.....	74
5.6	Classification Methods Using Feature Descriptors.....	76
5.6.1	Bayesian Classification	76
5.6.2	Neural Networks	77
5.6.3	Support Vector Machines.....	77
5.6.4	Association Rule Mining.....	78
5.6.5	KSTAR Classification.....	78

5.7 Feature Descriptor and Classification Evaluation.....	78
5.7.1 Columbus Large Image Format (CLIF)	79
5.7.2 Army Research Lab CEGR Infrared Dataset	80
5.7.3 Army Research Lab IICO Visible Dataset	84
5.8 Feature Extraction and Classification Conclusion.....	87
CHAPTER 6 - TRACKING FOR SE.....	89
6.1 Object Tracking Introduction.....	89
6.2 Optical Flow Object Tracking	90
6.3 DRSURF Feature Tracking.....	92
6.3.1 DRSURF Feature Tracking Implementation.....	92
6.3.2 Feature Tracking Evaluation and Results.....	93
6.3.3 DRSURF Feature Tracking Conclusion.....	95
6.4 Object Tracking Conclusion	96
CHAPTER 7 - GPU IMPLEMENTATION OF THE SE FRAMEWORK.....	97
7.1 GPU Introduction.....	97
7.2 GPU-Based SE Framework Implementations	98
7.2.1 GPU for Image Preprocessing.....	98
7.2.2 GPU for Object Detection	100
7.2.3 GPU for Feature Extraction	106
7.2.4 GPU for Object Tracking	113
7.2.5 GPU for Implementation Summary	117
7.3 GPU-Based SE Framework Conclusion	118
CHAPTER 8 - CONCLUSIONS	120

8.1 Contributions to Object Detection	120
8.2 Contributions to Feature Extraction and Classification	121
8.3 Contributions to Object Tracking	122
REFERENCES	124

LIST OF TABLES

Table 7.1 Algorithm GPU Speed-Up for SE	118
Table 7.2 Iterative GPU Speed-Up for SE.....	118

LIST OF FIGURES

Figure 1.1	SE System Step Diagram.....	2
Figure 1.2	Knowledge Discovery Steps In Data Mining.....	5
Figure 1.3	Data Mining For SE Steps.....	6
Figure 2.1	SE Framework Tasks and Methodologies.....	11
Figure 3.1	Image Processing Stage in SE Framework.....	22
Figure 3.2	Matrix Representation of an 2-D Image and Object Chips.....	23
Figure 3.3	Raw Image Object Chip in Grayscale and Jet Colormap, and Object Chip after Wiener Filter	25
Figure 3.4	Histogram Specification to Reduce Illumination Effects.....	26
Figure 3.5	<i>(Top)</i> Visible Aerial Object Chip Wavelet Decomposition Approximation Coefficients Levels 0-3 and <i>(Bottom)</i> Levels 1-6 on an IR Chip.....	29
Figure 3.6	<i>(Top)</i> 2-D Gaussian Filtering With Changes in Histogram, <i>(Bottom)</i> Original Object Chip, its Level 2 Wavelet Approximation, and Post Gaussian Filter Output.....	31
Figure 4.1	Object Detection in The SE Framework	32
Figure 4.2	<i>(Top left to right)</i> Image Frame With Multiple Object Motions and Wavelet Approximation $N=0\dots3$, <i>(2nd from top)</i> Lucas and Kanade Optical Flow for Each Wavelet Level, <i>(3rd from top)</i> Lucas and Kanade Pyramid Optical Flow for Each Wavelet Level, and <i>(Bottom)</i> Phase- Based Optical Flow Method.....	39
Figure 4.3	Computation Time at Each Wavelet Approximation Level $N=0,1,2,3$	40
Figure 4.4	Detection Accuracy Results for Each Optical Flow at Wavelet Levels $N=0,1,2,3$	41
Figure 4.5	Diagram of the Multi-Scale Graph Segmentation Algorithm [41].....	42

Figure 4.6	(<i>Left</i>) Multi-Scale Segmentation of Level 1 Approximation Image (from Figure 2) at Levels $g = 4,7,10$, (<i>right</i>) Interpolation Matrices ($Pg \rightarrow (g + 1)$) For Levels $g = 0,3,6,9$	45
Figure 4.7	Multiscale Graph Segmentations of a SAR Object Chip	46
Figure 4.8	Graph Segmentation Detection Accuracy of Objects in CLIF Dataset.....	50
Figure 4.9	Graph Segmentation of CLIF Objects Computation Time.....	50
Figure 4.10	(<i>Top left</i>) Original Image and (<i>top right</i>) its 2-Level 2-D Wavelet Approximation Coefficients (<i>bottom left</i>) Overlaid with Optical Flow Field. The Corresponding Flow Gradient Image is Shown (<i>bottom left</i>), and The Final Image after a 2-D Gaussian Filter is Applied (<i>bottom right</i>).....	53
Figure 4.11	Edge Gradient Flow Process	55
Figure 4.12	Hierarchical Node Agglomeration Using Multi-Cue	57
Figure 4.13	Binary Classification and Non-Maximal Suppression Results of Object Candidates	61
Figure 4.14	Object Detections in CLIF Data Sequences	61
Figure 4.15	ROC Curves of Binary Object Classification.....	62
Figure 4.16	Classifier Execution Speeds for Varying Object Candidate Sizes	63
Figure 5.1	Feature Extraction and Classification in the SE Framework.....	65
Figure 5.2	DRSURF Interest Point Location and Feature Window	67
Figure 5.3	DRSURF Grid Over Multiple Orientation	68
Figure 5.4	Scaled and Rotated Images and their Corresponding Invariant Moments	72
Figure 5.5	Canny Filtered Objects at Varying Wavelet Levels	73
Figure 5.6	Edge Bin Histogram of an Object Image	74
Figure 5.7	Steps in Texture Calculation	75
Figure 5.8	CLIF Object Detection Accuracy Using Edge Histograms.....	80
Figure 5.9	CEGR Classification Results Using Moment Invariants.....	82

Figure 5.10 CEGR Classification Results Using Edge Histogram	82
Figure 5.11 CEGR Classification Results Using Texture Statistics	83
Figure 5.12 CEGR Classification Results Using Invariant Moments, Edge Histograms, and Texture Statistics	83
Figure 5.13 IICO Classification Results Using Moment Invariants	85
Figure 5.14 IICO Classification Results Using Edge Histogram Statistics	86
Figure 5.15 IICO Classification Results Using Texture Statistics	86
Figure 5.16 IICO Classification Results Using Invariant Moments, Edge Histograms, and Texture Statistics at Wavelet Levels $N = 0,1,2,3,4,5, ALL$	87
Figure 6.1 Object Tracking in SE Framework	89
Figure 6.2 Object Tracking Accuracy Results for Each Optical Flow at Wavelet Levels $N=0,1,2,3$	91
Figure 6.3 DRSURF Feature Tracking Accuracy for Varying Feature Lengths	94
Figure 6.4 DRSURF Feature Tracking Execution Time for Varying Feature Lengths ..	94
Figure 6.5 Tracking Accuracy Results for DRSURF and Other Methods at Different Optical Flow Thresholds	95
Figure 7.1 GPU-CPU Execution Times for 2-D Gaussian Convolutions	99
Figure 7.2 Speed-Up Gains Using GPU for Gaussian 2-D Convolutions	100
Figure 7.3 GPU-CPU Execution Times for Gabor Filter Bank	101
Figure 7.4 GPU-CPU Speed Up for Gabor Filter Bank	101
Figure 7.5 GPU-CPU Execution Times for Graph Edge Weighting	102
Figure 7.6 GPU-CPU Speed Up for Graph Edge Weighting	103
Figure 7.7 GPU-CPU Execution Times for Graph Cuts (no loops)	104
Figure 7.8 GPU-CPU Speed-Up for Graph Cuts (no loops)	104
Figure 7.9 GPU-CPU Execution Times for Graph Cuts (small loops)	105

Figure 7.10 GPU-CPU Speed-Up for Graph Cuts (small loops).....	105
Figure 7.11 GPU-CPU Execution Times for Graph Cuts (large loops)	106
Figure 7.12 GPU-CPU Speed-Up for Graph Cuts (Large loops)	106
Figure 7.13 GPU-CPU Execution Time for Histogram (no loops)	107
Figure 7.14 GPU-CPU Speed-Up for Histogram (no loops)	108
Figure 7.15 GPU-CPU Execution Time for Histogram (small loops).....	108
Figure 7.16 GPU-CPU Speed-Up for Histogram (small loops)	109
Figure 7.17 GPU-CPU Execution Time for Histogram (large loops)	109
Figure 7.18 GPU-CPU Speed-Up for Histogram (large loops)	110
Figure 7.19 GPU-CPU Execution Time for Invariant Moments (no loops).....	111
Figure 7.20 GPU-CPU Speed-Up for Invariant Moments (no loops)	111
Figure 7.21 GPU-CPU Execution Time for Invariant Moments (small loops)	112
Figure 7.22 GPU-CPU Speed-Up for Invariant Moments (small loops).....	112
Figure 7.23 GPU-CPU Execution Time for Invariant Moments (large loops).....	113
Figure 7.24 GPU-CPU Speed-Up for Invariant Moments (large loops)	113
Figure 7.25 GPU-CPU Execution Time for Normalized Cross Correlation (no loops) .	114
Figure 7.26 GPU-CPU Speed-Up for Normalized Cross Correlation (no loops).....	115
Figure 7.27 GPU-CPU Execution Time for Normalized Cross Correlation (small loops)	115
Figure 7.28 GPU-CPU Speed-Up for Normalized Cross Correlation (small loops)	116
Figure 7.29 GPU-CPU Execution Time for Normalized Cross Correlation (large loops)	116
Figure 7.30 GPU-CPU Speed-Up for Normalized Cross Correlation (large loops).....	117

ACKNOWLEDGEMENTS

I would like first like to thank my advisor, Dr. Sumeet Dua, for his guidance, insight, and patience throughout my years as a graduate student. I would also like to acknowledge the Clarkson Aerospace and the Air Force Research Laboratory, especially Olga Mendoza-Schrock and Greg Arnold, for their support of the research.

I would also like to acknowledge the members of the Louisiana Tech Data Mining Research Laboratory, especially Harry, Brandy, Pradeep, Xian, Justin, and Connor for their daily assistance and friendship. I would also like to thank my parents, Brenda and Steve, for their love and support (not just financial) throughout the years of my graduate studies. Also, I want to thank my brothers, Stephen and Kevin, and friends for always being there despite my sometimes long absence from their daily lives. Finally, I want to thank Brandie for her love, patience, and support throughout the past years of the degree completion. I love you, Sweetie.

CHAPTER 1

INTRODUCTION

1.1 What is Object Recognition?

Over the past several decades, many industries have invested greatly into research and implementation of performing autonomous surveillance with increasing complexity. The reasoning behind moving towards automation in systems is three-fold: reducing human casualties, human error, and reducing the financial cost of human operation. A particular field of surveillance that can benefit from automation is in sensor exploitation (SE), which outputs a knowledge layer using sensor data as input.

An SE system effectively removes man from the process of the acquisition, classification, and tracking objects of interest. As object-mounted sensor technologies exponentially increase coverage areas, coupled with booming available computational resources, human-run objecting systems can no longer keep up with potential object acquisitions in real time scenarios. Multiple sensor modalities exist concurrently on objects, along with other data such as GPS and satellite imagery to provide a rich, layered sensing stack that a single human operator cannot manage unassisted. SE provides the required assistance to aid human operators in utilizing the sensing capabilities currently available in the specific platform.

In its most basic form, SE acquires (detects), classifies, and tracks objects by processing a sequences of images [1]. This process can be broken down into a series of sub-tasks, which take the output from a previous step as input into the current step. Figure 1.1 describes the typical algorithm subtask sequence in an SE application, each of which will be described in further detail.

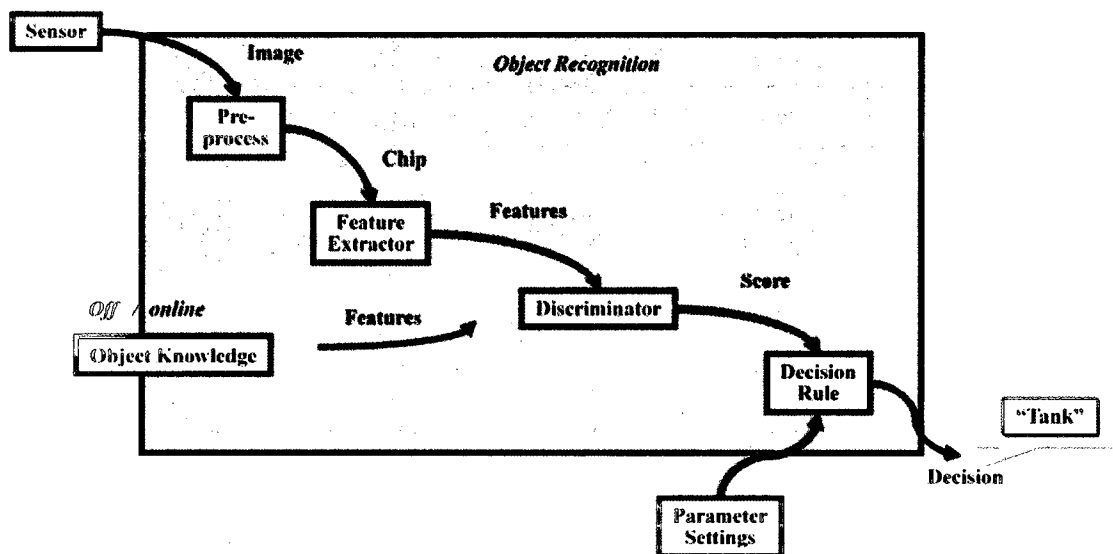


Figure 1.1 SE System Step Diagram

The SE method begins with sensor input data at the top left, termed “Sensor,” which is typically one or several images. The input images are then fed into the SE system as follow:

1. **Pre-screener:** This initial stage conditions the input image into a form acceptable for feature extraction methods. Typically, subtasks such as image denoising, data reduction, and initial non-object filtering are performed to both enhance speed and accuracy of later stages. A set of potential object “chips” is outputted from this

stage, where each chip represents a location (bounding box or boundary) from the input image(s).

2. **Feature Extractor:** This stage takes a potential object chip and calculates a single or set of values that can discriminate a member of an object library from non-objects (confusers). Feature extraction methods are domain-specific, due to the specific requirements of invariance to object changes (rotational, translation, scale, partial occlusion, illumination, etc.)
3. **Discriminator:** This step uses either an online or offline training system and features from the previous step to attain an object “likeliness” score, which can either be binary or probability based. The likeliness score for each object chip is then passed to the decision rule stage.
4. **Decision Rule:** This final stage uses likeliness scores and parameter settings to discretely label potential object chips as either “non-object” or “object” in a binary detection system, or a particular object in a multi-object detection. Both online information (neighboring likeliness scores) and offline (user-defined thresholds of likeliness) are used to enhance accuracy, and adjust the false-alarm and false-dismissal ratios.

SE suffers from a lack of robust solutions due to several clear challenges in the problem space. Such challenges include a large range of potential input sensor data, changing operating conditions (OCs), cluttered environments, object signature non-repeatability, limited or incomplete testing data, and real-time execution [1]. Many of these issues also occur in problems in data mining, which is the driving force for choosing a data mining-based solution to the SE problems.

1.2 Data Mining and the Knowledge Discovery Process

The diverse applications of data mining and general knowledge discovery in databases (KDD) causes a lack of agreement on the basic definitions of the field [2]. Although it is unclear whether KDD and data mining are one in the same, or merely a step in the KDD process. As with SE, knowledge discovery can be described as a series of sequential steps:

1. **Data Processing (Cleaning):** Raw data will contain noisy and missing entries which must be resolved to decrease potential misclassification. This stage typically involves several processing steps that resolve particular irregularities present in the data domain.
2. **Data Reduction:** Data dimensionality size can inhibit mining methods by both sheer computational burden and reduce accuracy from redundant, meaningless data. Reduction methods are employed to retain only useful information through compression or filtering methods.
3. **Data Transformation:** Processed, reduced data is then transformed using mathematical techniques for exposing discriminative characteristics of the data. The transformations are chosen based on input structure requirements for the classification or decision making stage.
4. **Data Mining:** In this crucial step, discriminative patterns from training data are used with statistical classification schemes to generate relationships, anomalies, trends, and patterns using the transformed data.
5. **Pattern Verification and Evaluation:** This final stage evaluates data mining methods by comparing predictions based on testing and training data used to

design the model for higher-level pattern detection and classification. This step provides visual representation of the accuracy and effectiveness of the data mining method.

Figure 1.2 is a diagram of the KDD process, which shows the typical algorithmic workflow. The SE and KDD processes have significant overlap in the overall methodologies, both beginning with large data that contains a significantly smaller set of desired information (object data). Ideas from both methods are combined in the framework, presenting an SE framework that includes ideas and workflow from the KDD process.

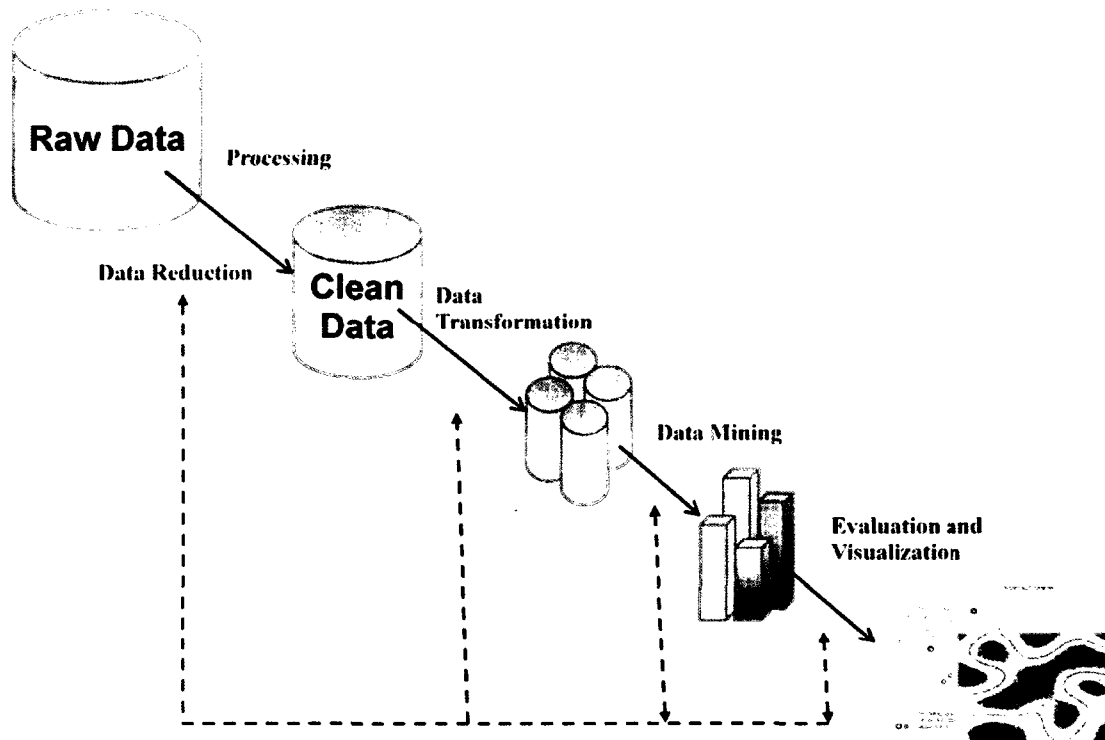


Figure 1.2 Knowledge Discovery Steps in Data Mining

1.3 Data Mining-Based SE Method

As previously described, the KDD and SE steps require similar sequence of subtasks, due to the strong coherence in their problem space. Figure 1.3 contains the data mining steps for SE that contains both critical stages in a comprehensive SE and KDD process. This approach to solving the SE problems provides the guidance to develop the SE framework described in Figure 1.3.

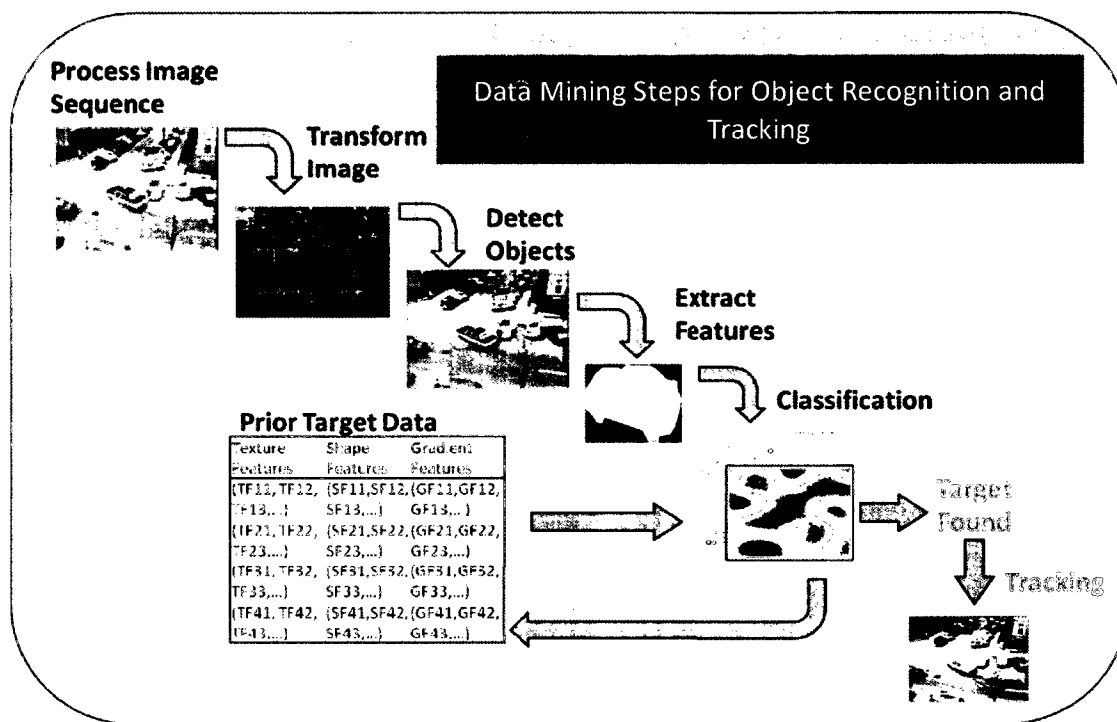


Figure 1.3 Data Mining for SE Steps

1. **Image Sequence Processing:** Raw images acquired from either ground or moving sensors will frequently contain noise generated from both the acquisition process (camera or transmission) and also from OCs such as non-uniform illumination and natural phenomena. Denoising operations are performed on the

input data to reduce these irregularities in aims to diminish further propagations throughout the process, resulting in inaccuracies. Non-transform data reductions can also be applied in this stage for filtering out obvious non-object locations.

2. **Data Transformation:** This next stage transforms cleaned and filtered image data into a form that can then be used to extract discriminative information for accurate detection and classification. This stage can also reduce data dimensionality through compression transformations, eradicating redundant information and increasing computation time.
3. **Object Detection:** This critical SE stage discriminates objects from non-objects using a wide array of possible methods. This method in itself can be broken down into a data mining process when a detection is the ultimate SE aim. This stage can also be considered an initial detection stage but retains false alarms (filtering), which are then reduced in classification stages. Object detection is the focus of several chapters of this dissertation, and several multiple approaches to object detection are presented.
4. **Feature Extraction:** This step describes the variety of statistical calculations used to compactly represent a region of interest (ROI), which in turn is passed to a classification or decision rule algorithm for object discrimination. Feature extraction methods must take account for the domain-specific intra-class and inter-class variations that can potentially arise, requiring invariant features to best represent these changing characteristics.
5. **Classification:** Taking feature sets and training data (object priors) as input, this stage uses both supervised machine learning methods or statistical analysis to

determine the ultimate class or label of a particular region of interest. Common classes can be binary, such as “object” or “non-object,” or can consist of an object library where multiple object classes exist.

6. **Tracking:** This final stage of SE updates the position of a correctly classified object (detected). Changes in object orientation and scale, along with partial occlusions requires a tracker to be robust to changes in object appearance. Although some trackers can be considered “detection” algorithms in that the object is re-detected and associated to itself at different time intervals, the tracking stage is considered independent from object detection.

These stages lay the groundwork for the SE framework presented in this paper, some of which demand multi-stage algorithms within each particular step. Although data mining steps can describe the entire SE system, steps 3,4, and 6 can each be considered an independent data mining problem, in cases where other stages are not required.

1.4 Dissertation Outline

In Chapter 2, the SE framework that solves each of the SE problems is discussed in the introduction, along with background on current methods employed. In Chapter 3, the initial image processing and data transformation methods used to condition the data for further use in subsequent steps are described. Next, Chapter 4 will detail the object detection methods used for both moving and non-moving object recognition, using phase-based optical flow and hierarchical graph segmentation with supervised classification. These two methods are then combined with machine classification for detecting moving objects through multiple frames. In Chapter 5, novel feature extraction methods used for binary and multiclass object classification are presented, and compared to several

machine-learning methods. In Chapter 6, a novel feature tracking method that employs existing features (from previous steps) for real-time tracking results is employed. In Chapter 7, the GPU implementations of the algorithms described in Chapters 3-7 are presented and discuss execution time gains. Finally, the conclusions and potential future research directions are presented in Chapter 8.

CHAPTER 2

SENSOR EXPLOITATION FRAMEWORK

2.1 SE Framework Introduction

The SE framework can be conceptualized as a series of necessary computer vision tasks that uses both supervised and unsupervised learning methods to detect, classify and track both moving and non-moving objects throughout an image sequence. The algorithm is based on the sequence design on the data mining-based SE diagram described in Section 1.3, creating and implementing a set of specific sub-tasks for each step.

Figure 2.1 provides a flow chart-style diagram of the required tasks and the methods employed to accomplish them. Most of these tasks have been outlined in Chapter 1, with additional sub-tasks specific to the image-based SE challenges. These methods are used to reduce false alarms and speed completion time, increasing performance at each downstream step. In this chapter, the background research performed to accomplish the tasks is outlined in Figure 2.1, and a brief introduction into each of the methods developed for solving the particular task is discussed. Each major task depicted in a blue box is a main topic in subsequent chapters.

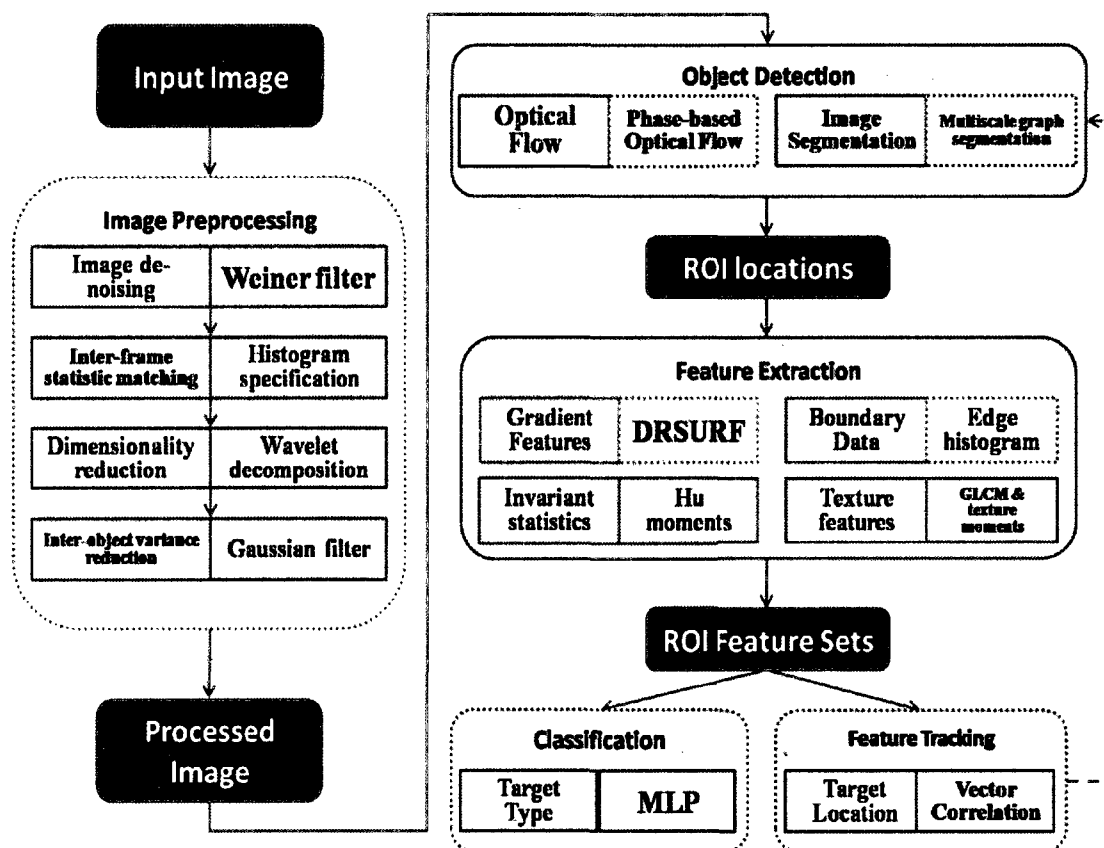


Figure 2.1 SE Framework Tasks and Methodologies

2.2 Image Preprocessing

An initial stage within most higher-level computer vision processes are one or more image preprocessing steps to condition image pixel values for increased robustness and accuracy. Typical preprocessing steps include image denoising, background elimination, contrast enhancement, and dimensionality reduction. Several tasks and their accompanying methods that are necessary for generating accurate detection, tracking, and classification results are described.

1. **Image denoising (smoothing)** describes the set of processes that attempts to remove degradations from image data caused by instrument imperfections, data

acquisition, interfering natural phenomena, transmission errors and compression [3]. Image denoising techniques can be broken up into two basic approaches: spatial filtering methods and transfer domain methods. Spatial filters can use both linear and nonlinear low-pass kernels, which convolve with an image to smooth or blur, at the cost of reducing object boundary contrast. Nonlinear methods such as weighted median [4] and relaxed median filtering [5] help combat this drawback. Transform domain filtering methods include spatial-frequency filters such as Fast Fourier Transform (FFT), wavelet transform, independent component analysis. FFT methods first transform the data into the frequency domain, then uses a frequency domain filter and adaptive cut-off frequency [6]. Wavelet methods create a set of multi-resolution coefficients which are then used for linear [7] and non-linear [8][9] filtering, coefficient modeling [10][11], and non-orthogonal transform methods[12][13]. Independent component analysis (ICA) assumes a signal is non-gaussian, but requires a higher computational cost [14]. A two-dimensional linear spatial filter, the Wiener filter, is implemented. This filter provides denoising at low computational cost and minimal boundary contrast reduction. At the end of the preprocessing step, a second linear smoothing step is also performed. This step uses a Gaussian kernel to blur processed images to homogenize intra-object values for increased detection accuracy.

2. **Inter-frame statistical matching** helps eliminate illumination changes when performing tasks that require multiple image region matching and comparisons, such as object tracking and motion estimation. Histogram transformation

methods treat image intensity distribution as a probability density function (pdf), which is then used to fit a specific distribution [15]. Histogram equalization both increases contrast and normalizes an image sequence through matching the intensity histogram to equal-depth bins [16]. Histogram specification uses a reference model histogram to transform an image's histogram to match that of the model most closely [15]. Specification is used to retain denoised image statistics and most closely match global statistics from previous images in a sequence.

3. **Dimensionality reduction** methods decrease image size to retain only informative data for further processes, as well as minimizing computation time. Methods that can achieve this task include data transformations such as principal component analysis (PCA) [17], fast Fourier transform (FFT) [18], and wavelet transformation [19]. PCA is an orthogonal linear transformation, which reduces a set of variables by calculating eigenvalues from a covariance matrix, retaining values with most contrast (information). FFT coefficients can be thresholded by frequency to reduce high frequency data. Wavelet transform allows for a multiscale depiction of the image. Unlike Fourier descriptors, wavelet coefficients retain spatial information because their basis functions (or wavelets) are localized in the image, where a Fourier basis function spans the entire image. Wavelet approximation (low-pass) coefficients are used to represent images in reduced dimensions. Different scales are used throughout the process but are all calculated efficiently in this step.
4. **Inter-object variance reduction** is required to take into account differences in intra-object variations between objects within a single class. An example of such

a case is the boundaries that occur between a light colored object and its windshield, which is absent for darker colored objects. This additional smoothing step creates a local blurring to reduce these sharp contrasts. Background modeling has been proposed which can reduce contrast [20] over a localized area. A simple 2-D Gaussian kernel function, as described in [15], is used due to its computational speed and effectiveness in reducing the overall quality of results.

2.3 Object Detection

An important process in the SE framework, the object detection strategy uses multiple techniques to both detect moving and non-moving regions of interest which conform to predefined size, shape, and motion characteristics. The methods output regions of interest locations that are then used as input for feature extraction, tracking, and classification methods.

1. **Optical Flow** computes an approximation of the 2-D motion field from spatiotemporal patterns of image intensities, which is a projection of the 3-D velocities of surface points into the image surface [21]. Motion estimation in image sequences via optical flow methods was first introduced by Horn and Schunk in 1980-81 [22]. Since this introduction, new techniques have been developed that calculate 2-D velocity fields using a variety of approaches. In the survey paper from Barron [23], the author selects a local intensity-based method by Lucas and Kanade [24] and a phase-based approach by Fleet and Jepson [25] as providing results with the highest accuracy and density. After evaluating these methods, a phase-based approach for motion estimation using wavelet approximation is performed.

2. **Image Segmentation** of non-trivial images is one of the most difficult tasks in image processing. Accurate segmentation typically leads to the success or failure of the analysis process [15]. Although the amount of image segmentation methods and research is immense, most methods can be divided into detecting discontinuities and similarities in an image region. Discontinuity detection is typical for methods that find *edges* in images through kernel operations [26], which are then followed with morphological operations to create connected component regions [27]. To segment regions that display edge occlusion, edge detection and linkage methods will fail in many scenarios. Similarity measures, on the other hand, are robust to occlusion because they tend to search over a local region for image statistics that display similar values. Simple methods that use similarity are intensity and color thresholding, with more advanced methods using histogram statistics in local regions for better discrimination [28]. Region-based methods can employ both pixel intensity similarities and discontinuities for segmentation such as watersheds [29] or region-growing methods that use seed points [30]. Graph-theory is a popular method for creating partitions and connections between similar pixels, forming regions. The normalized cuts [31] method uses similarity measures to “cut” graph node connections while keeping strong connections together, while other methods use shape information to create segments from the graph [32][33]. Multi-scale segmentation has proven successful in finding relationships between intra-segment statistics at different scales. Such methods include model-based approaches [34] and probabilistic linking [35]. Many methods employ wavelet decompositions for segmentation,

such as using a multi-scale hidden Markov tree [36] and neural-networks [37]. A novel implementation of the multi-scale graph theoretic segmentation using wavelet coefficients for object detection is used.

3. **Track-before-detect** is a popular detection method that when applied to high frame rate data, provides high detection accuracy over multiple image frames [38]. The method uses a similar approach in terms of requiring matching of multiple instances of a detected object to reduce false alarms, but instead of using a dynamic state space model technique such as [39] and [40], object matching and non-maximal suppression are employed to discriminate object candidates in the subsequent frame. Both optical flow and graph segmentation methods are employed to acquire the short tracks to validate detected objects. These acquired motion features are then combined with intensity and edge features to form a multi-cue energy function in a hierarchical graph segmentation algorithm inspired by [41].

2.4 Feature Extraction and Classification

In many domains that require data mining methods for classification, proper feature set selection is a critical step in attaining accurate results. For SE, robust feature selection is especially difficult due to the many scale, rotation, illumination, and environmental variations that occur within a single image sequence. Several classes of feature extraction techniques are surveyed and novel extensions of them that have produced accurate classification results are created.

1. **Gradient feature descriptors** are a general class of features describes some of the most recent and highly celebrated methods to date. These methods use first

and second order derivatives to identify unique local areas that can best be differentiated using matching algorithms in subsequent frames or in object libraries. Corner point detection methods, such as Harris corner points [42] and differential invariants [43], provide a means to find locations for calculating descriptors that can be tracked through a series of frames. Mikolajczyk and Schmid [44] provide a thorough evaluation of local descriptors for image matching (object tracking), observing SIFT (Scale Invariant Feature Transform) [45], and speed-up robust features (SURF) [46] and GLOH (Gradient Location and Orientation Histogram) methods as ranking highest in performance. Histogram and gradients (HOG) are used in the classification stage [47]. Steerable filters can also provide rotational invariance by “steering” in a particular direction based on the highest gradient [48].

2. **Boundary and shape descriptors** use an object silhouette, or outer boundary, to describe the object for discrimination. Typically a boundary detection method, like [26], must be employed at some part in the feature extraction process, which uses contrast with the background to detect discontinuities. Log-polar shape descriptors are presented in [49] to detect aerial objects. Belongie et al introduce “shape contexts” for matching similar shapes to one another [50]. Non-linear shape statistics have also been used for segmentation and tracking by using a density estimation method [51]. Boundary information typically lacks rotational invariance, thus, a feature set is created that both maintains shape discrimination and invariance to rotation.

3. **Invariant descriptors** is a large research area that handles the multiple variations that occur in SE is a primary challenge. Invariant feature extraction is a large research area, which includes some very common methods such as histograms [52] and invariant moments [53], which provide invariant feature sets to rotation, translation, and scaling changes. Recent extensions include the geometric histogram [54] and scale-invariant descriptors [55]. Both invariant features and several types of histograms for classification are used. These forms of classification can help to detect objects with high variations due to rotations and partial occlusions.
4. **Texture-based descriptors** belong to a class that uses the internal statistics of an object for classification, referred to as “texture.” Such methods include calculating gray-level co-occurrence matrices [15], then generating the celebrated Haralick features [56]. Some methods use filters such as the Gabor transform [57] and wavelets [58] to generate a set of texture classifiers. Due to the large variations within classes of objects, a very limited set of texture features are used, although the feature set is very important in other image recognition fields.

In the methodology, many of the calculated feature sets are used as inputs into supervised classification algorithms. Previous image classification and SE research is dominated by supervised classification schemes. In [59], spatial and texture features are calculated and used in a two-stage classification algorithm, which uses both rule-based classification for object detection, then linear discriminate analysis (LDA) and quadratic discriminate analysis to (QDA) to further classify objects types. Cao et al. [60] employ support vector machines and wavelet moment invariants to classify SAR images into a 3-

class object classification system. Neural-networks directed Bayes-decision rules creates conditional probabilities for classification, using object motion characteristics as the primary features [61].

2.5 Object Tracking

In this final stage of the SE framework, previous knowledge derived from the detection and classification stages is used to provide a feature tracking scheme that generates a probability of a current object location. Object tracking is determining the trajectory of an object throughout a 2-D or 3-D surface. This undertaking is challenging due to: loss of information due to 2-D projection of a 3-D scene, image noise, complex motion, non-rigid objects, partial or full occlusions, complex shapes, environmental conditions, and real-time processing requirements [62]. The object tracking research is rich in methods that employ supervised classification, dynamic state space models, feature matching, and iterative detections. The background of each of these methods will be described in more detail.

1. **Tracking by classification** uses online training of classifiers to track through an image sequence by labeling a location as an “object” class and all non-object locations as “non-objects.” Avidan uses support vector machines (SVM) with an optical flow algorithm to maximize a classification score to update the object location [63]. Williams et al. use full probabilistic relevance vector machine (RVM) to generate observation with Gaussian distributions to track by determining the displacement of the object [64]. Bayesian networks have also been successfully implemented for object tracking [65].

2. **Dynamic state models** use ideas from control theory, dynamic state modeling for object tracking is a very popular method with current object tracking research. The dynamic state of the changing position is governed by the dynamic equation, which includes previous states and noise. Methods based on classical dynamic modeling, such as Kalman filters [66][67] and particle filters [68][69] are used for their robustness to changing operating conditions. These methods can retain object tracks over partial and full occlusions, although they can suffer from issues of drifting.
3. **Feature matching** is a successful class of tracking algorithms which locate unique features associated with spatial areas over multiple frames. Methods such as the KLT tracker use features discussed in Section 2.4 to match similar feature sets to generate similarity scores [70]. Cross-correlation is a basic method for determining the similarity between two sets of image descriptors, which is used as a similarity metric in Chapter 6. Distance measures are another means to determine the matching measurement, with low distances equating to strong matches. Multiple tracking methods use the Mahalanobis distance metric, while other tracking methods have used covariance matrices, homography estimations, and Euclidean distances [42]. These distance measures can affect tracking results through feature sensitivity, as discussed in the methods section.
4. **Iterative detection (detect-before-track)** performs a detection using a clustering or segmentation to determine object presence, which is associated with a particular track. Such methods include using optical flow and segmentation (described in Section 2.3), with the additional detection and tracking methods

such as mean-shift [71] and Track-and-cut [72]. These detect-before-track methods can suffer from heavier computation, but can benefit from performing independent segmentation tasks in parallel.

Background into each of the classes of object tracking methodologies is discussed, and the vast areas of research exploration within each of these areas are identified. Although a feature matching-based method (described in Chapter 6) is currently used, several ideas are derived from other tracking methods throughout the SE framework.

2.6 SE Framework Description Conclusion

This chapter serves as a brief introduction into each of the SE framework stages, providing the reader with a glimpse into the wide range of research areas involved in creating solutions to the challenges that SE design faces. Each of these stages can easily garner investigation and exploration that could produce dissertation-level research. The main aims of this dissertation are to demonstrate how each of these areas of research can be combined into a working set of algorithms to achieve the entire set of SE challenges in a single framework. Each of the algorithms and SE challenges is described in detail throughout the next several chapters.

CHAPTER 3

IMAGE PREPROCESSING

3.1 Introduction

As discussed in Chapter 2, an initial stage of the series of tasks are performed to condition an input image or images prior to higher-level operations. In Figure 3.1, the subroutines highlighted in red display the sequential image processing stage.

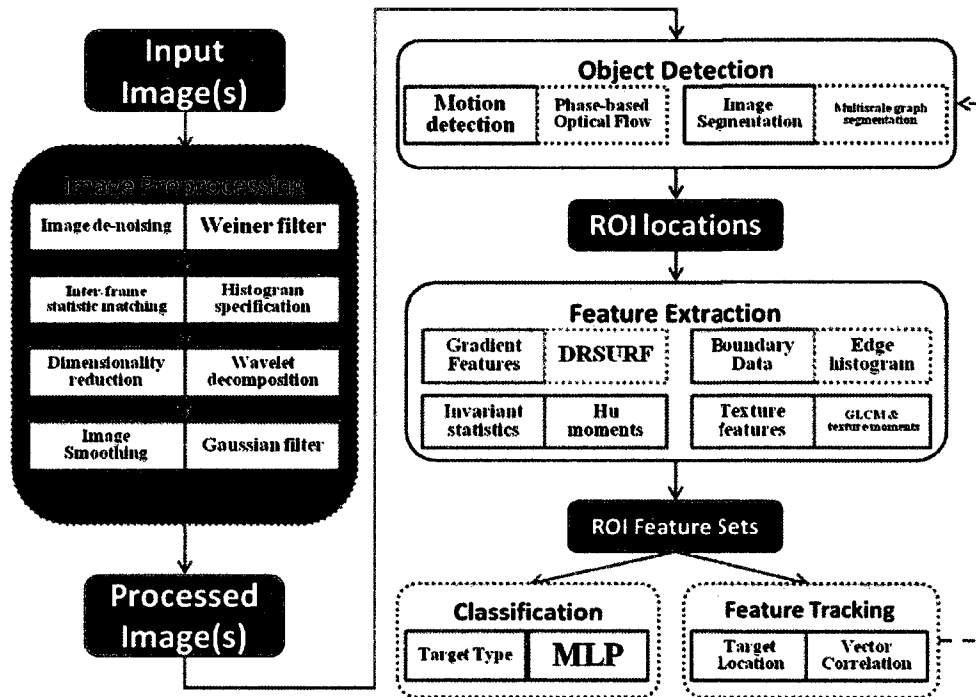


Figure 3.1 Image Processing Stage in SE Framework

First, the digital sensor image representation is described as a discrete function, $f(x, y)$, of pixel size $M \times N$. For this description, the discussion will be limited to grayscale, or intensity images, represented as:

$$f(x, y) = I(x, y). \quad (3.1)$$

For each (x, y) location (*note in matrix form, x refers to rows and y refers to columns*). These intensities make up a two-dimensional matrix, with ranges for 8-bit images between $[0, 255]$, or if normalized between $[0, 1]$. These intensities can also be thought of as magnitudes at a particular location (Figure 3.2), allowing to more easily visualize many of the computational methods discussed in the rest of the chapters. Also shown in Figure 3.2 are typical image sensor data in visible and infrared spectrums.

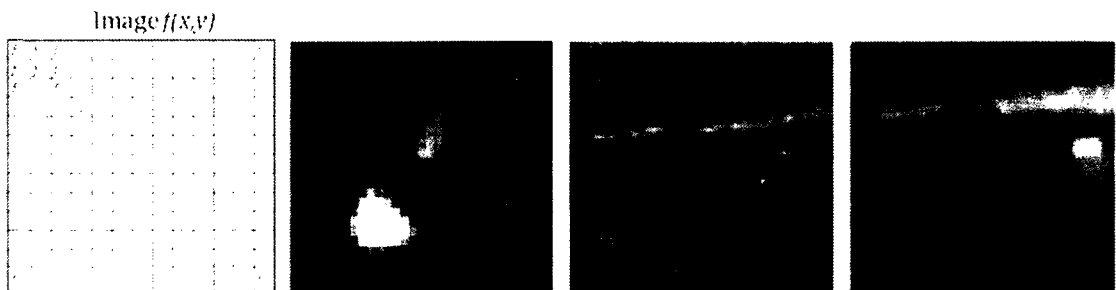


Figure 3.2 Matrix Representation of an 2-D Image and Object Chips

As explained in both Chapters 1 and 2, raw input data must be conditioned for successful information exploitation. Image sequence exploitation is no different, with sensor data series requiring denoising, data reduction and filtering techniques to provide higher order algorithms with appropriately “cleaned” input data to produce accurate results.

Several methods have been employed in the SE framework, each of which serves a unique purpose down the chain of algorithms. Although these methods are not the only ones capable of denoising, reducing, transforming, and filtering data sets for SE, they have been selected for both their specific use for the data sets of interest and because they can easily be augmented for data sets with differing properties.

3.2 The Wiener Filter

The Wiener filter is applied to adaptively reduce noise artifacts. Using a local neighborhood of size $m \times n$, an average of the local estimated variances, v , is used to create a pixel-wise filter:

$$J(i, j) = \mu + \frac{\sigma^2 + v^2}{\sigma^2} (I(i, j) - \mu), \quad (3.2)$$

where μ is the local mean, σ the local standard deviation, and $I(i, j)$ is the image pixel value at location (i, j) . In the implementation, the local neighborhood is set to $n = m = 3$, but can easily be automated by using prior knowledge of object dimensions. Figure 3.3, shows the original raw intensity object chip of an aerial object in grayscale (left), then display the same object chip in the center using jet colormap (middle) for better visualization. The last image is the output after performing a 2-D Wiener filter with the mean calculated over a 5×5 window.

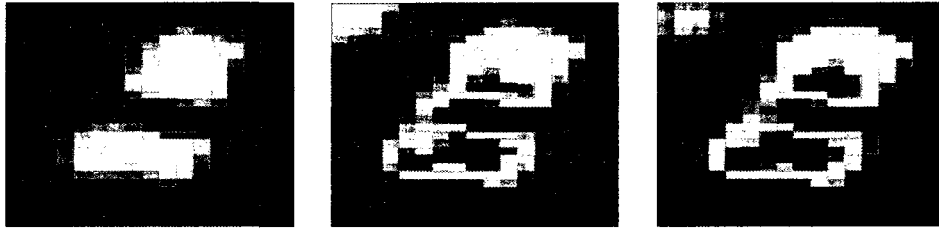


Figure 3.3 Raw Image Object Chip in Grayscale and Jet Colormap, and Object Chip after Wiener Filter

The Wiener filtering reduces small, local variations that can mislead edge detection and gradient feature extraction methods. Notice how both background and object local areas smoothen within its segment, with little to no blurring of boundary areas. This function is first performed so that noisy signals do not propagate into forms that cannot be easily cleaned, such as after data reduction steps.

3.3 Histogram Specification

Now that local noise has been reduced through Wiener filtering, additional image irregularities can occur which cannot be solved through calculations of localized areas, such as environmental changes due to illumination. To combat such issues, the next preprocessing step is to perform histogram specification [15] between the subsequent images used for the input. Histogram specification approximately matches the intensity histogram from image I_t to the histogram from image I_{t+dt} by creating a transform function which maps the probability density function (pdf) from the histogram I_t to the histogram I_{t+dt} .

For discrete pixel values, the probability of occurrence of intensity level r_k in an image is approximated by:

$$p_r(r_k) = \frac{n_k}{n} \quad k = 0, 1, 2, \dots, L - 1, \quad (3.4)$$

where n is the total pixel count of the entire image, n_k is the count of pixels that have intensity r_k , and L is the range of intensity values (0-255 for 8-bit images). The histogram equalization transform function is then:

$$\begin{aligned} s_k = T(r_k) &= \sum_{j=0}^k p_r(r_j) \\ &= \sum_{j=0}^k \frac{n_j}{n} \quad k = 0, 1, 2, \dots, L - 1 \end{aligned} \quad (3.5)$$

Each image pixel r_k is mapped to s_k through transform T , resulting in forming an image with a histogram that is much more evenly spread out across the range $[0 \dots L-1]$ of pixel intensities. This step is performed to reduce false optical flow values from illumination changes between image frames. In Figure 3.4, you can see the subtle changes in the histogram, which performs an important role of reducing illumination effects that result in poor motion estimation.

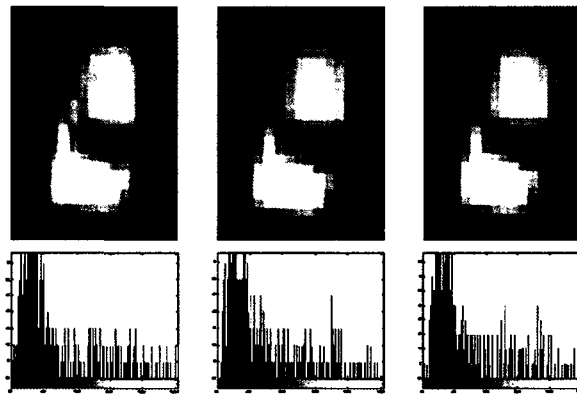


Figure 3.4 Histogram Specification to Reduce Illumination Effects

3.4 Wavelet Decomposition

Wavelet decomposition has been employed for the purpose of dimensionality and data redundancy reduction, providing a multiscale solution-space for detection scale-specific motion within an image frame [74]. Before implementing the detection algorithms, a 2-D discrete wavelet transform of the processed image is performed to create a multiscale representation of the image frame. The 2-D discrete wavelet transform uses a wavelet function, $\psi(x, y)$, and a scaling function, $\varphi(x, y)$ to correctly position the function before convolving with the image of size $M \times N$ using the following formula:

$$\varphi_{j,m,n}(x, y) = 2^{j/2}\varphi(2^jx - m, 2^jy - n), \quad (3.6)$$

$$\psi^i_{j,m,n}(x, y) = 2^{j/2}\psi^i(2^jx - m, 2^jy - n), \quad i = \{H, V, D\} \quad (3.7)$$

The 2-D discrete wavelet transform of the function $f(x, y)$ of an image of size $M \times N$ is formalized as [19]:

$$W_\varphi(j_0, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \varphi_{j_0,m,n}(x, y), \quad (3.8)$$

$$W_\psi^i(j, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \psi^i_{j,m,n}(x, y) \quad i = \{H, V, D\}, \quad (3.9)$$

where j_0 is the starting scale, which is typically set to $j_0 = 0$, then set to $N = M = 2^j$

and $j = 0, 1, 2, \dots, J - 1$ and $m, n = 0, 1, 2, \dots, 2^j - 1$, and $\{H, V, D\}$ yield the 2-D

combination to get ψ^i for a horizontal (H), vertical (V), and diagonal (D) direction.

Daubechies-3 and Haar wavelets are used, as they have evidenced to offer high degrees of detection and tracking accuracy in previous studies [75]. The generated wavelet approximation coefficients, W_φ , are used as input into the optical flow algorithm

described in Chapter 4. Due to the 2^j scaling relationship between pixel locations, optical flow fields can be easily interpolated to find flow at original image locations. The first three levels of wavelet approximation coefficients are found, with higher scales detecting larger motions. The notation W^L to refer to a wavelet approximation of an image, I , at level L is used.

In Figure 3.5, the approximation coefficients of wavelet decompositions are shown on visible and IR sensor data. As can be seen from Figure 3.5, object data sizes are reduced but characteristics can still be retained at lower wavelet levels. Based on the resolution of the image, high scale approximations lose any resemblance to the object chip it represents, which in most cases is an indication the coefficients cannot be used to successfully discriminate the object from the background.

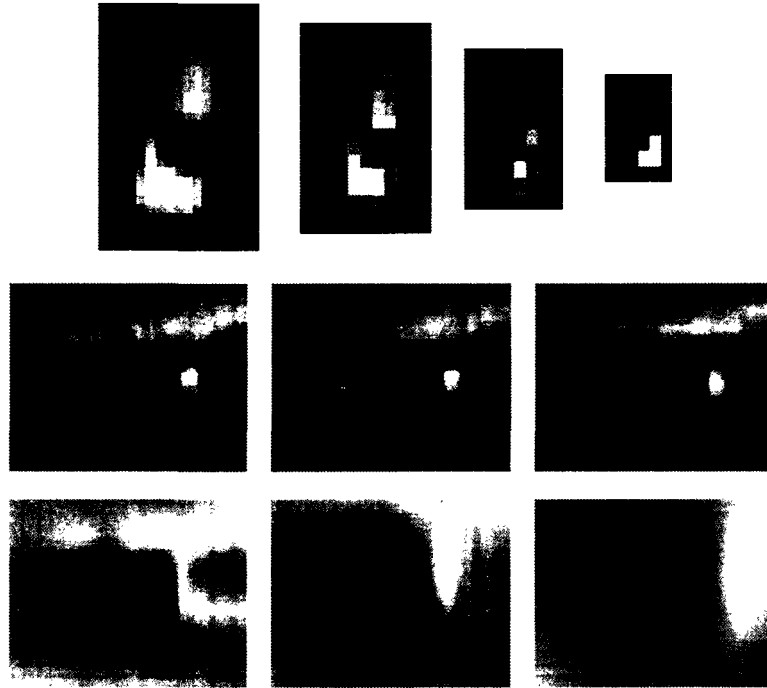


Figure 3.5 (*Top*) Visible Aerial Object Chip Wavelet Decomposition Approximation Coefficients Levels 0-3 and (*bottom*) Levels 1-6 on an IR Chip

3.5 Gaussian Smoothing

Gaussian smoothing was implemented in the final stage of preprocessing. Intra-object non-uniform pixel variations can make image segmentation difficult because areas that contain sub-segments of the same object may differ too significantly to merge together. This second noise reduction filter also assists motion detection by decreasing discontinuities that do not appear in subsequent frames due to occlusion. The size of the kernel was chosen to be roughly $\frac{1}{4}$ the size of the smallest object to both retain object edge information and sufficiently smooth intra-object pixel intensities.

The Gaussian kernel is a member of a class of spatial filters; spatial filters are used in preprocessing stages to reduce noise and help connect edge and object regions through

combining neighborhood pixels to transform a central pixel intensity or color value. The Gaussian kernel $g(x, y)$ is formalized as:

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}, \quad (3.10)$$

where σ determines the width of the kernel. A smoothing mask or kernel is a matrix of coefficients (usually rectangular or circular) which is convolved with an image. The coefficients increase toward the center of the kernel to give larger weight to pixel locations closest to the central pixel of the passing image window. Figure 3.6 show the changes to an object that occur after a Gaussian filter is applied. Take note that some boundary information is lost through this smoothing process, unlike the Wiener filter. This loss in boundary localization is acceptable in cases where large intra-class variations are significantly reduced.

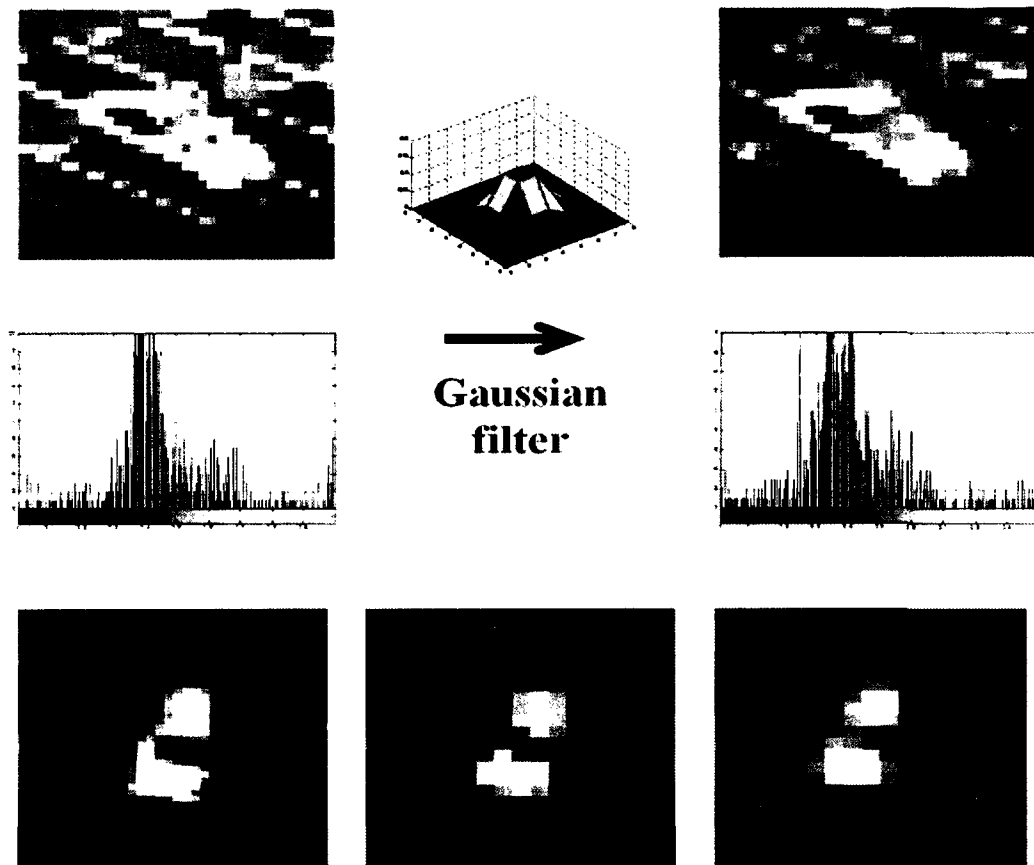


Figure 3.6 (*Top*) 2-D Gaussian Filtering With Changes in Histogram, (*bottom*) Original Object Chip, its Level 2 Wavelet Approximation, and Post Gaussian Filter Output

3.6 Conclusion

The image preprocessing methodology that was performed to condition the data for the subsequent SE stages follows. Image preprocessing methods are very data-specific, with this particular set of methods providing acceptable results in the evaluation. It would be advantageous to experiment with several preprocessing methods when attempting to perform SE on a dataset of sensor data that does not conform to the characteristics of previous tested data.

CHAPTER 4

OBJECT DETECTION

4.1 Introduction

Now that image data has been conditioned using the methods described in the previous chapter, the critical stage of object detection takes the processed data as input, then performs multiple algorithms that detect motion and segments images to then determine potential object locations (Figure 4.1 highlighted in red).

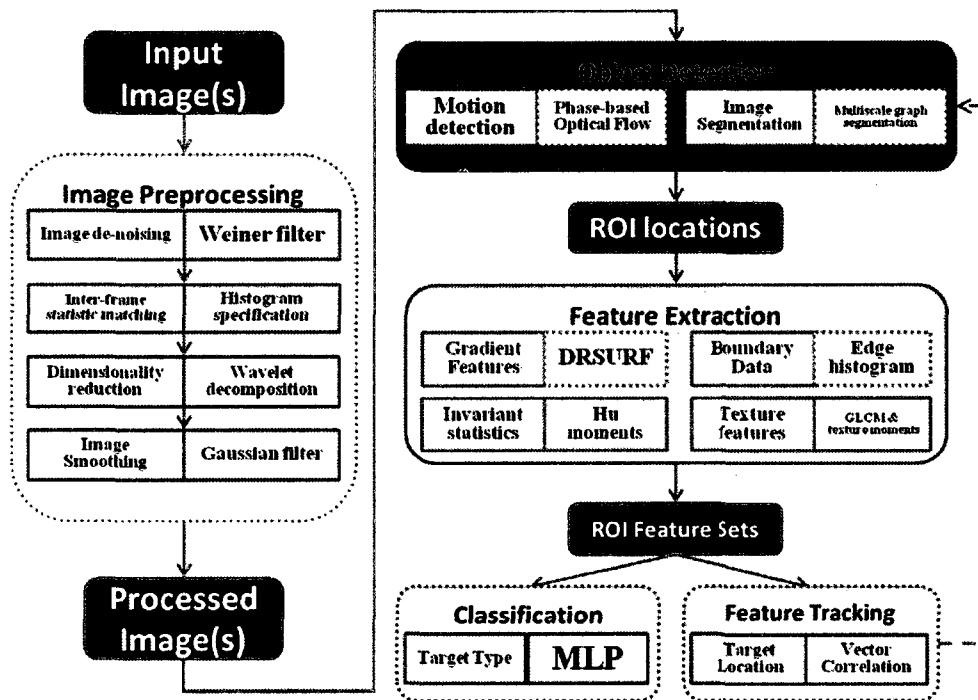


Figure 4.1 Object Detection in the SE Framework

This process requires sophisticated algorithms that demand potential object prior data, due to the myriad of possible ways to break down even a simple real image into discrete parts. In the subsequent sections of this chapter, the use of exploit sensor data to detect object motion is explained, then create a hierarchical segmentation of the scene to extract only segments which match domain knowledge of object candidates are explained. A full object detection methodology that reduces false alarms by employing a track-before-detect system is presented. Evaluations of each of the methods are also presented and discussed.

4.2 Motion Detection Using Optical Flow

As discussed in Chapter 2, two optical flow calculation methods, one intensity-based and one phase-based, have been chosen to evaluate wavelet approximations of image scenes for object detection and tracking. In the following sections, each method is described, along with an explanation about how they differ in calculating the optical flow values.

4.2.1 Lucas and Kanade Method

In the Lucas and Kanade optical flow method, an iterative approach is used to find velocity components for images $A = I(t)$ and $B = I(t + dt)$. The image velocity, $\vec{v} = [v_x \ v_y]^T$, can be defined as the vector that minimizes the residual function ε defined at image position $\mathbf{p} = [p_x \ p_y]^T$ as [76]:

$$\varepsilon(\vec{v}) = \varepsilon(v_x, v_y) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x, y) - B(x + v_x, y + v_y))^2. \quad (4.1)$$

An integration window, ω , is chosen to have a size both large enough to find large object motions and small enough to retain local accuracy. Using prior domain knowledge of object velocity can allow for the automatic selection of ω to the smallest window that can contain maximum object motion. An optimum optical flow calculation will give:

$$\left. \frac{\partial \varepsilon(\bar{v})}{\partial v} \right|_{\bar{v}=\bar{v}_{\text{opt}}} = [0 \ 0]. \quad (4.2)$$

After expanding Equation (4.2) from Equation (4.1):

$$\frac{\partial \varepsilon(\bar{v})}{\partial v} = -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x, y) - B(x + v_x, y + v_y)) \cdot \left[\frac{\partial B}{\partial x} \ \frac{\partial B}{\partial y} \right]. \quad (4.3)$$

After substituting $B(x + v_x, y + v_y)$ by its first order Taylor expansion for the point $\bar{v} = [0 \ 0]^T$, it can be seen that the $(A(x, y) - B(x, y))$ is the image gradient between the two images, with the image gradient vector written as:

$$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \doteq A(x, y) - B(x, y). \quad (4.4)$$

The derivative images I_x and I_y can be found from image A only through the use of a central difference operator. Equation (4.3) can then be rewritten using the difference images as:

$$\frac{1}{2} \left[\frac{\partial \varepsilon(\bar{v})}{\partial v} \right]^T \approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \bar{v} - \begin{bmatrix} \delta I \ I_x \\ \delta I \ I_y \end{bmatrix} \right). \quad (4.5)$$

Denote

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad (4.6)$$

$$\bar{b} \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I \ I_x \\ \delta I \ I_y \end{bmatrix}. \quad (4.7)$$

Now, Equation 4.2 can be rewritten as:

$$\frac{1}{2} \left[\frac{\partial \varepsilon(\bar{v})}{\partial v} \right]^T \approx G \bar{v} - \bar{b}. \quad (4.8)$$

The optimum optical flow vector is then:

$$\bar{v}_{\text{opt}} \approx G^{-1} \bar{b}. \quad (4.9)$$

This equation is only a valid result when G is invertible, which occurs when the gradients in image A are nonzero in both the x and y directions. This method is used iteratively through a Newton-Raphson method to find larger pixel displacements. The algorithm will perform k iterations before stopping. These k iterations improve optical flow accuracy, but require additional computation time. This algorithm is implemented using multiple levels ($N = 0,1,2,3$) of wavelet approximations as inputs and set the iterations to $k = 3$.

4.2.2 Pyramid Implementation of Lucas and Kanade

This multi-scale approach to the Lucas and Kanade optical flow calculation is motivated by the possibility of calculating motion at both large and small scales by recursively reducing the image size while keeping the window size constant. The pyramidal representation of the input image I of size $M \times N$ will be denoted as I^0 , or the zeroth level of the image with dimensions $m^0 = M$ and $n^0 = N$. The pyramid is then recursively constructed using a Taylor series expansion around the each point $I^{L-1}(2x, 2y)$ to get I^L . This expansion is possible due to the coordinate relationship between levels in L as:

$$(x, y)_L = \frac{(x, y)_0}{2^L}. \quad (4.10)$$

The image at I^{L-1} is defined as follows (through Taylor series expansion):

$$\begin{aligned}
 I^L(x, y) = & \frac{1}{4} I^{L-1}(2x, 2y) \\
 & + \frac{1}{8} [I^{L-1}(2x - 1, 2y) + I^{L-1}(2x + 1, 2y) + I^{L-1}(2x, 2y - 1) \\
 & + I^{L-1}(2x, 2y + 1)] \\
 & + \frac{1}{16} [I^{L-1}(2x - 1, 2y - 1) + I^{L-1}(2x + 1, 2y + 1) \\
 & + I^{L-1}(2x - 1, 2y + 1) + I^{L-1}(2x + 1, 2y - 1)].
 \end{aligned}$$

Similar to wavelet approximation coefficients, the image pyramid-level height is typically $L_{max} = 2, 3, 4$, because the large reduction in size makes levels > 3 useless for determining local motion. The pyramidal Lucas Kanade using the same parameters as from Section 4.2.1. is then implemented, setting the additional pyramid parameter level to 3 to ensure multi-scale optical flow calculations.

4.2.3 Phase-Based Optical Flow

For a third estimate of motion, a phase-based optical flow method from Gautama and Van Hulle, which uses spatial Gabor filters for velocity estimation [77], is used. This method, similar to that Fleet and Jepson [25], tracks contours of constant phase over time, which have been proven to be more robust to illumination changes and non-translation motions than contours of constant amplitude. This method is divided into three stages.

This method begins by filtering the image multiple times using quadrature pairs of Gabor filters, where a temporal phase gradient is computed. Each filter is characterized by its center frequency, (f_x, f_y) , and the width, σ , of the Gaussian curve which produces a complex-valued phase response, with a phase component $\phi(x, t)$. As Fleet and Jepson state, the temporal changes of a constant phase can approximate motion, which is

represented as contours that satisfy $\phi(x, t) = c$. This equation is differentiated with respect to t , which gives us the temporal phase gradient, $\nabla \phi(x, t)$:

$$\nabla \phi(x, t) \cdot \nabla x = \nabla \phi(x, t) \cdot (v, 1), \quad (4.11)$$

where v is the velocity vector. The temporal phase gradient can then be written in the following manner:

$$\phi_t = -(\mathbf{v} \cdot \phi_x) = -\|\phi_x\| \text{proj}_{\phi_x^n}(\mathbf{v}), \quad (4.12)$$

where ϕ_x^n is the normalized ϕ_x . The component velocity in the direction normal to the filter orientation can now be computed as:

$$v_c = \text{proj}_{\phi_x^n}(\mathbf{v}) \phi_x^n = \frac{-\phi_t(x)}{2\pi(f_x^2 + f_y^2)} (f_x, f_y). \quad (4.13)$$

In the second stage of the algorithm, the reliability of the component velocities is measured, due to inaccurate velocities computed at phase singularities. The *linearity* of the component velocities are measured by performing a linear least squares regression on the phase component pairs, taking the mean square error divided by the gradient to yield phase nonlinearity. High non-linearity values above a defined threshold are rejected. The linearity threshold is then chosen as, $thresh_{lin} = .05$, as suggested by the authors to reject phases between $[-150, 150]$, which begins producing unreliable results.

The final stage of the algorithm is the combination of reliable component velocities into a single velocity measure at each location in the x and y directions. The following equation is used to create the full velocity using a constraint line, L_i , yielding the full velocity at the point where several constraint lines intersect. Guatama and Van Hulle use a goal programming method that uses amplifiers to converge to a solution for a full velocity, $\mathbf{v} = (u, v)$ from component velocities by minimizing the distances between

constraint lines. An iteration step of the goal programming network is written as [78]:

$$v^{k+1} = v^k - \Delta s \sum_{i=1}^N v_{c,i} \left(\frac{v^k \cdot v_{c,i}}{\|v_{c,i}\|} - \|v_{c,i}\| \right), \quad (4.14)$$

where N is the number of constraint lines and Δs is the time interval between state updates. This algorithm is implemented using a Gabor filter bank of size 11 filter pairs using $\beta = 1$ to get a Gaussian width, σ , using the following equation:

$$\sigma = \frac{2^{\beta+1}}{(2^{\beta}-1)2\pi\sqrt{f_x^2+f_y^2}}. \quad (4.15)$$

The minimum number of constraint lines was set to $N = 5$ for all of the implementations. The next stage of the algorithm processes and reshapes optical flow calculations to match original image sequence dimensions. The optical flow for each wavelet approximation level, j , will have the dimension size relationship of $M_j \times N_j = M/2^j \times N/2^j$, with an interpolation necessary to map values to the original image size. The wavelet scaling function is used to interpolate the optical flow values to mimic the compression of the image (Section 3.3). The values of the rescaled optical flow are then multiplied at each location by 2^j to account for the increase in pixel distances. Figure 4.2 provides typical optical flow results for wavelet approximation levels $N = 0,1,2,3$ for each of the three methods described in Sections 4.2.1-4.2.3.

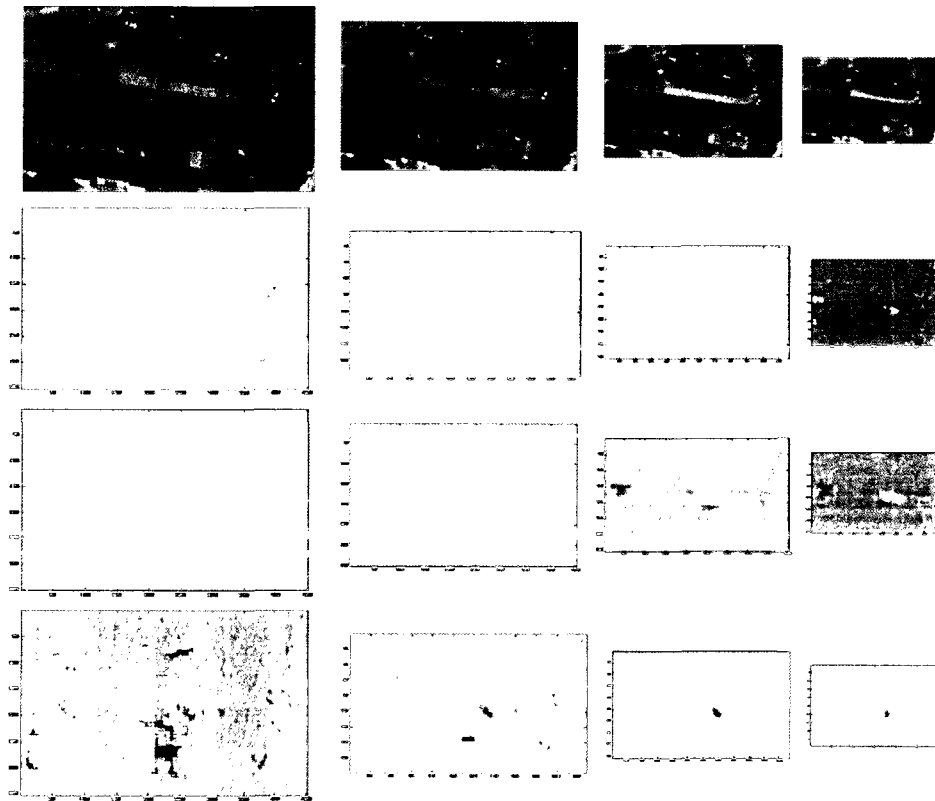


Figure 4.2 (*Top left to right*) Image Frame With Multiple Object Motions and Wavelet Approximation $N=0\dots3$, (*2nd from top*) Lucas and Kanade Optical Flow for Each Wavelet Level, (*3rd from top*) Lucas and Kanade Pyramid Optical Flow for Each Wavelet Level, and (*bottom*) Phase-Based Optical Flow Method.

4.2.4 Evaluation and Results

For evaluating the optical flow calculations, a subset of the camera 1 Columbus Large Image Format (CLIF) sequence [79] has been chosen. This camera captures an aerial view of an urban area at a two frame/sec rate. Because the image sequence dimensions are too large for complete processing (9733 x 8033), sub-image sequences of varying sizes between 700 x 700 and 214 x 529 have been created. In total, six sequences were used ranging between 20 and 38 frames in each and containing 16 object ground truth data. The optical flow calculations were computed using Matlab R2009a release on an

Intel® Core i7 920 @ 2.67 GHz processor with 12 GB of RAM. The completion time for each wavelet level and optical flow technique is provided in Figure 4.3.

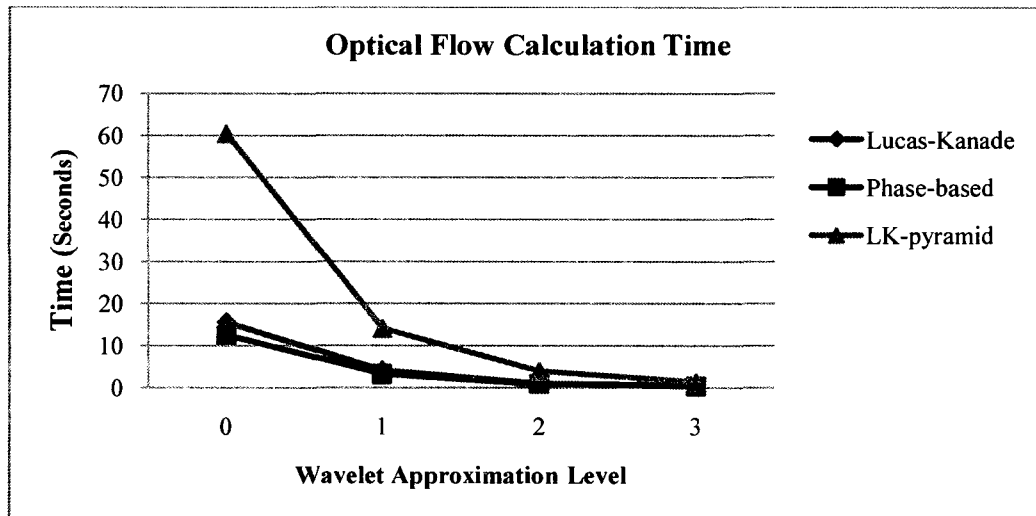


Figure 4.3 Computation Time at Each Wavelet Approximation Level $N=0,1,2,3$.

Object detection at a specific pixel location is based on pixel locations where

$|\mu_{target\ n_{max}}| > thresh_v$, with the threshold chosen for each optical flow method based on prior knowledge of minimum optical flow values from a moving object. The results are provided in Figure 4.4, which shows object detection accuracy and overall detection area. Optimum performance gives high detection accuracy with low detection area.

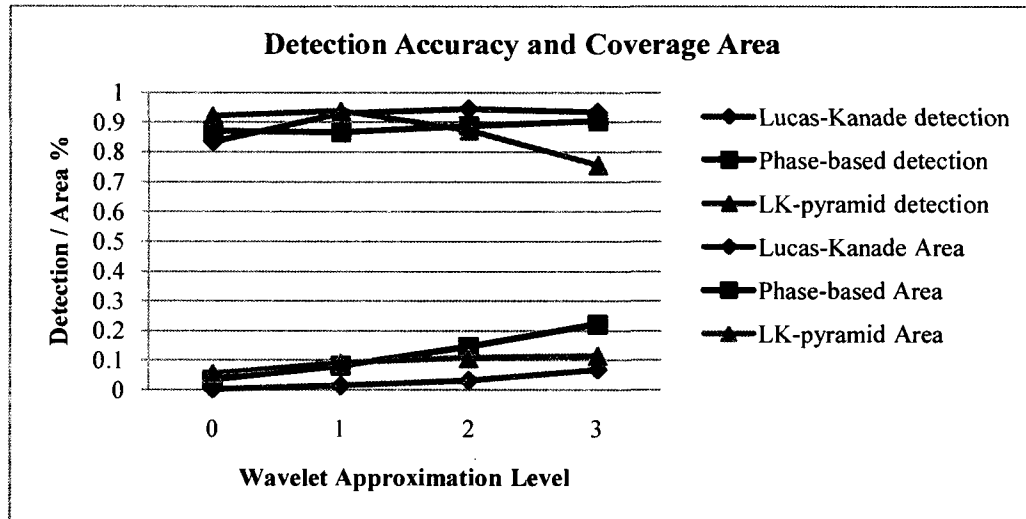


Figure 4.4 Detection Accuracy Results for Each Optical Flow at Wavelet Levels $N=0,1,2,3$.

4.2.5 Optical Flow Evaluation Conclusions

Although all three optical flow methods were successful in detecting object motion, the phase-based method achieved superior object tracking accuracy (91.05% at level-0), possibly due to the illumination invariance that phase-based methods achieve. Wavelet approximations provide an exponential decrease in computational time, which was expected due to the dimensionality reduction. Detection accuracy and specificity are reduced with the increase in wavelet decomposition levels, and the best compromise is the phase-based method at level 1 (87% accuracy). Although computational time is still too large for use in a real-time analysis, lower-level language implementation coupled with parallel programming can reduce this computation time significantly.

4.3 Object Detection Using Hierarchical Graph Segmentation

This method uses the concept of multi-scale graph segmentation for the bottom-up aggregation of nodes, allowing image areas to merge together at different scales, similar

to methods described in [80][41]. As outlined in Figure 4.5, the method begins with a construction of a level 0 graph using intensity contrast as weights for edges. An iterative graph coarsening method is then used to aggregate nodes at a new scale. This step is repeated until a predetermined minimum amount of nodes has been reached. Higher-level weighting functions are used to augment weights, which use node statistics to change node couplings. After the algorithm completes, the graphs are scanned for segments similar to desired object shape and size.

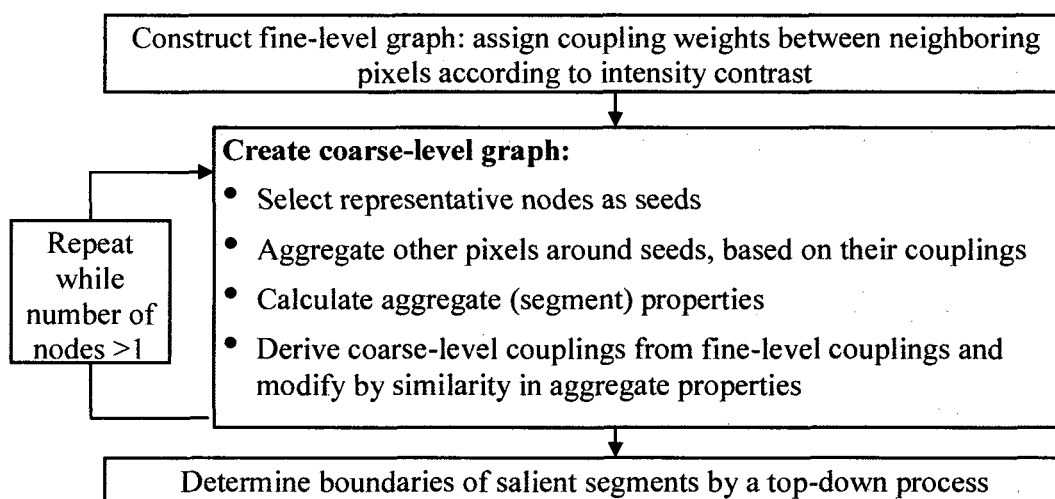


Figure 4.5 Diagram of the Multi-Scale Graph Segmentation Algorithm [41]

The multi-scale graph-theoretic segmentation algorithm is divided into its constituent parts and discuss its implementation. The algorithm subsections include: (Section 4.3.1) graph initialization, (Section 4.3.2) energy function for node selection, (Section 4.3.3) graph coarsening, (Section 4.3.4) inter-node weighting using node statistics, and (Section 4.3.5) object segmentation candidate selection.

4.3.1 Graph Initialization

In the initial stage (level 0) of the graph segmentation algorithm, each pixel in image I of size $M \times N$ is considered as a node in a graph, with an edge connecting each 4-connected neighbor. An inclusion matrix, U_g , (g is graph scale) is created which is used to find pixel membership ($U > 0$) at a particular level. The initial membership matrix U_0 of size $(M \times N) \times (M \times N)$ has a value of:

$$U_0(i, j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad \text{for } i, j = 1, 2, \dots, M * N, \quad (4.16)$$

so that each row in U_0 has only one member at location $i = j$, creating an identity matrix. A weight matrix, W_0 , of the same size as U_0 , is also created and populated with values from edge couplings. This edge (connection) strength is initially dependent solely on pixel intensity similarities, with high similarity creating stronger edge strength. The function to determine the initial edge is written as [80]:

$$w_{ij} = e^{-\alpha |I_i - I_j|}, \quad (4.17)$$

where i and j are nodes in a graph representing 4-connected neighboring pixels, I is image intensity value, and α is a global parameter set to 0.25 to best segment targets of desired size. The a list of edges ($W_0 > 0$) and node locations ($U_0 > 0$) are then used to create a Laplacian matrix, L_0 , which is used in the energy equation in Section 4.3.2.

4.3.2 Node Energy Function

To decide which nodes are chosen to be representative nodes in the next scale, ($g = 1$), an energy function, $\Gamma(u)$, is calculated using the Laplacian matrix, weight matrix, and inclusion matrix. Sharon et al. concluded that when solving the generalized eigen problem $Lu = \lambda Wu$ with minimal positive eigenvalue λ , salient nodes are found

where $\Gamma(\mathbf{u})$ is minimal. The following equation is then used to generate energy values for each node in the inclusion matrix [41]:

$$\Gamma(\mathbf{u}) = \frac{\sum_{i>j} w_{ij}(u_i - u_j)^2}{\sum_{i>j} w_{ij}u_iu_j} = \frac{\mathbf{u}^T L \mathbf{u}}{\frac{1}{2} \mathbf{u}^T W \mathbf{u}}. \quad (4.18)$$

The median value of Γ is chosen as the threshold in determining which nodes are selected as node locations at the next coarsest level. As nodes grow in size, an additional size constant, β , is introduced to reduce the tendency of larger nodes to be selected, changing Equation (4.18) as follows:

$$\Gamma(\mathbf{u}) = \frac{\mathbf{u}^T L \mathbf{u}}{\left(\frac{1}{2} \mathbf{u}^T W \mathbf{u}\right)^\beta}. \quad (4.19)$$

Setting $1 > \beta > .5$ will increase the tendency for larger nodes to be chosen as new coarse nodes, where $.5 > \beta > 0$ will cause smaller node selection to increase. $\beta = .5$ eliminates any size bias in the node selection. Depending on the dimensions of the image frame compared with the desired object dimension, β can be automatically selected to create nodes of a size similar to prior object dimensions.

4.3.3 Graph Coarsening

Candidate nodes are selected using the Γ function described above as all nodes that satisfy $\Gamma(\mathbf{u}) < \text{median}(\Gamma)$. The number of selected nodes is typically $\frac{\mathbf{u}}{2}$ in size. The inclusion matrix, for the new level U_{g+1} is created with an approximate size $N_g \times N_g/2$, where N_g is the number of nodes of the graph at level g . After the nodes with minimum energy are selected, an interpolation matrix, $P_{g \rightarrow (g+1)}$ is created. This new interpolation matrix uses weights from W_g to associate non-candidate nodes to candidate nodes. After

$P_{g \rightarrow (g+1)}$ is populated with weight values from non-candidate nodes, the nodes at level g are scanned again to find any nodes that were not selected as candidates and did not have connections to candidate nodes. Nodes in this pass are defined as size N_{g+1} . These “orphaned” nodes are appended to $P_{g \rightarrow (g+1)}$ and U_{g+1} , as additional candidate node locations. The interpolation matrix is then normalized so that:

$$\sum_{i=1}^{N_{g+1}} p_{ik} = 1 \quad \text{for all } k > N_{g+1}, \quad (4.20)$$

which gives a normalized weighted relationship for all nodes not selected as coarse level nodes. Using the interpolation matrix, the weights are determined at the node scale level $g + 1$ as:

$$W_{g+1} = P_{g \rightarrow (g+1)}^T W_g P_{g \rightarrow (g+1)}. \quad (4.21)$$

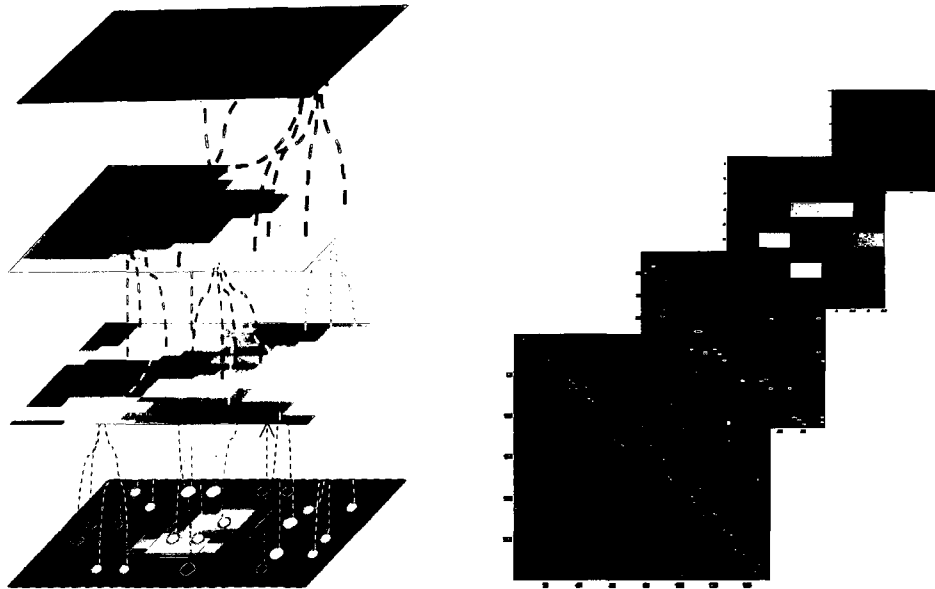


Figure 4.6 (Left) Multi-Scale Segmentation of Level 1 Approximation Image at Levels $g = 4, 7, 10$, (right) Interpolation Matrices ($P_{g \rightarrow (g+1)}$) For Levels $g = 0, 3, 6, 9$.

4.3.4 Higher-level Node Statistics

After initial weights (Section 4.3.1) and interpolation matrices (Section 4.3.3) augment the edge weights connecting nodes at levels $g > 0$, additional statistics are calculated, and their similarity measure is used to reduce or increase node weights. For most of these calculations, the original pixel membership in current node scale must be found. This membership can be found by multiplying interpolation matrices from the current scale down to $P_{0 \rightarrow 1}$. This step will create a matrix, V , of size $N_g \times N_0$. For each pixel, I (1 to N_0) is assigned to a single node by finding the max values in the row vector v_i . Each pixel ultimately becomes a member of a node at level g , which is used to find the node statistics described in Figure 4.7.

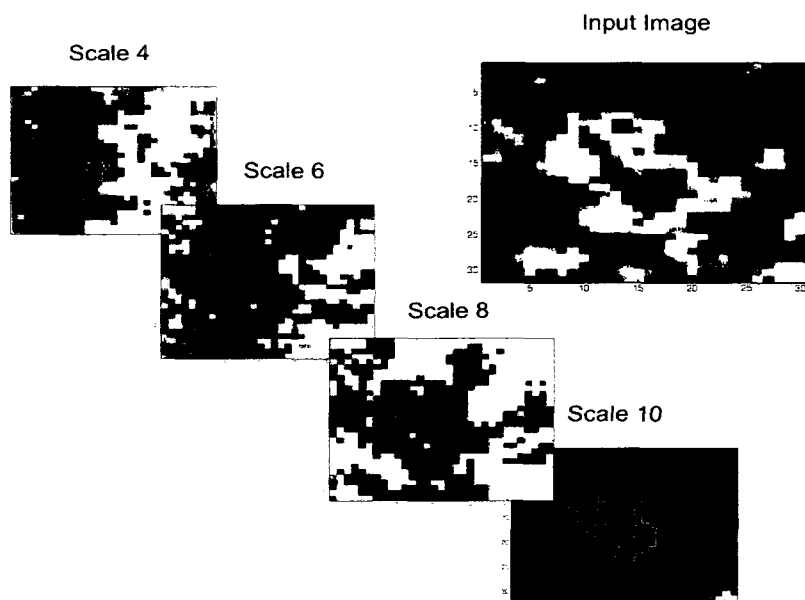


Figure 4.7 Multiscale Graph Segmentations of a SAR Object Chip

Once nodes represent multiple pixels, the V matrix using the interpolation matrices below level g can be used to find intensity statistics for node k and scale g , which are then used to find similar nodes at level g . After identifying members (pixels) of each k node at

level g , the pixel members can be described as an intensity histogram, which is written as a probability density function (pdf):

$$p_r(r_k) = \frac{n_k}{n} \quad k = 0, 1, 2, \dots, L - 1, \quad (4.22)$$

where $p_r(r_k)$ is the probability of an image pixel being a member of histogram bin r_k .

For this method, $k = 16$ bins is chosen in the texture calculations which provides the required textural discrimination, unless noted otherwise. To calculate the moments, write the moment equation as follows:

$$\mu_n(r) = \sum_{k=0}^{L-1} (r_k - m)^n p(r_k), \quad (4.23)$$

where the mean value of r is:

$$m = \sum_{k=0}^{L-1} r_k * p(r_k). \quad (4.24)$$

The second, third, and fourth moments, which are the *standard deviation*, *skewness*, and *kurtosis*, respectively are calculated. Then, this feature vector is compared to other nodes in scale g using a vector correlation function:

$$corr_{k,j} = \frac{[v_k] \cdot [v_j]}{\sqrt{([v_k] \cdot [v_k]) * ([v_j] \cdot [v_j])}}, \quad k, j = 1, 2, \dots, N_g. \quad (4.25)$$

The correlation value is used to augment existing weights between nodes by using the flowing function:

$$w_{kj} e^{-\alpha_{stats} |1 - corr_{k,j}|}, \quad (4.26)$$

where α_{stats} is a weighting constant specific for each weighting function used.

Correlations of values of $corr_{k,j} < 1$ will reduce coupling weights.

Another statistical measure of interest is the inter-scale variance at finer scales for each node. Unlike the intensity statistics, the multi-scale variance calculates the variance of nodes at particular scales below g . These variance values are discovered by finding V at each level below g , creating a vector of variances for each node k a level g of desired length. Variance is kept at levels $g - 1, g - 2, g - 3$. Variance vectors are then compared, and weights are augmented using Equations (4.25) and (4.26) with an α_{var} parameter.

A third measure used to augment node couplings at a scale g is boundary comparison. After each node's pixel members are found using V , boundary pixels are identified, and a list of non-node member neighbors is created. Then, node weights are increased between neighboring nodes from the list that satisfy a mean intensity threshold, τ_{mean} , written as:

$$w_{kj} = \alpha_{boundary} * w_{kj} \text{ if } |\mu_k - \mu_j| < \tau_{mean}, \quad (4.27)$$

where $\alpha_{Boundary}$ is a predefined weight constant, μ_k , and μ_j are intensity means of nodes k and j at level g found from these methods.

4.3.5 Object Candidate Selection

The final stage of the algorithm uses prior object knowledge to scan the segmentation at each scale and extract potential objects based on size and shape statistics. A high and low threshold, τ_{high} and τ_{low} , is then multiplied by the statistics of the largest and smallest prior object ground truths. In this case, τ_{high} is set to 1.5 and τ_{low} is set to .5. Total area is set to minimum and maximum axis length and eccentricity to find segments that fall between the high and low thresholds. Results for implementing this object candidate selection with the segmentation method are described in Section 4.3.6.

4.3.6 Segmentation Evaluation and Results

The purpose of using a segmentation method for object detection in an image frame for finding both moving and non-moving objects is that an object can be considered “detected” if a segmented region of interest (ROI) has been found within the distance $\tau_{centroid}$ from the centroid of the object ground truth. The parameters are set as $\alpha = .25, \beta = .5, \alpha_{stats} = .05, \alpha_{var} = .05, \alpha_{boundary} = 2, \tau_{mean} = 20$. The dataset has been tested on wavelet approximation $level = 0,1,2,3$. For testing segmentation accuracy for object detection, the same subset of the camera 0 Columbus Large Image Format (CLIF) sequence described in Section 4.2.4 has been chosen.

As can be seen in Figure 4.8, graph segmentation provides strong detection accuracy at each of the wavelet approximation levels. There is a slight decrease at wavelet-level 2, possibly due to object boundary shape and resolution. Figure 4.9 shows the segmentation time for image chips that include objects and surrounding neighborhoods (approx 50 x 50 pixels). As can be seen from Figure 4.9, computation time decreases logarithmically with an increase in wavelet level. This correlated decrease is due to the exponential reduction in dimensionality.

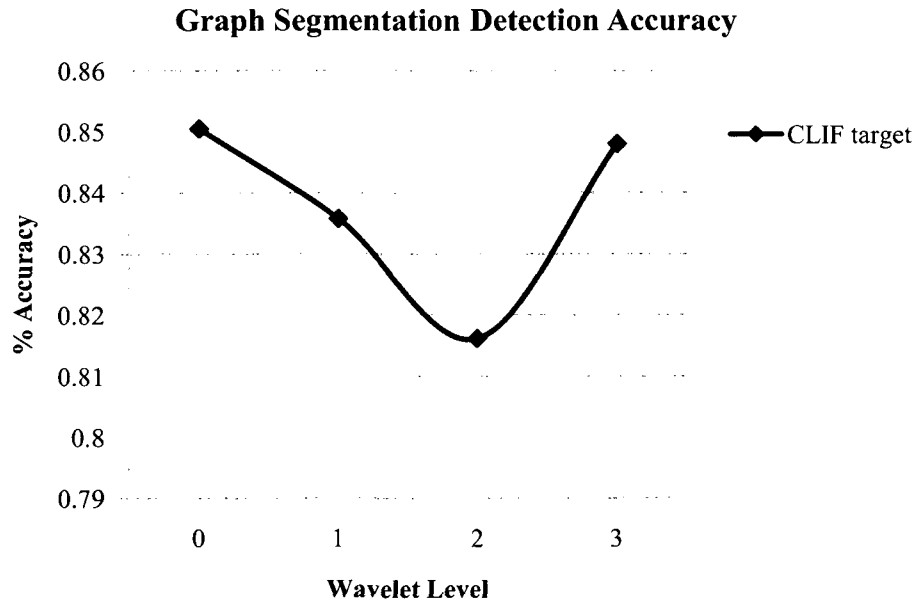


Figure 4.8 Graph Segmentation Detection Accuracy of Objects in CLIF Dataset

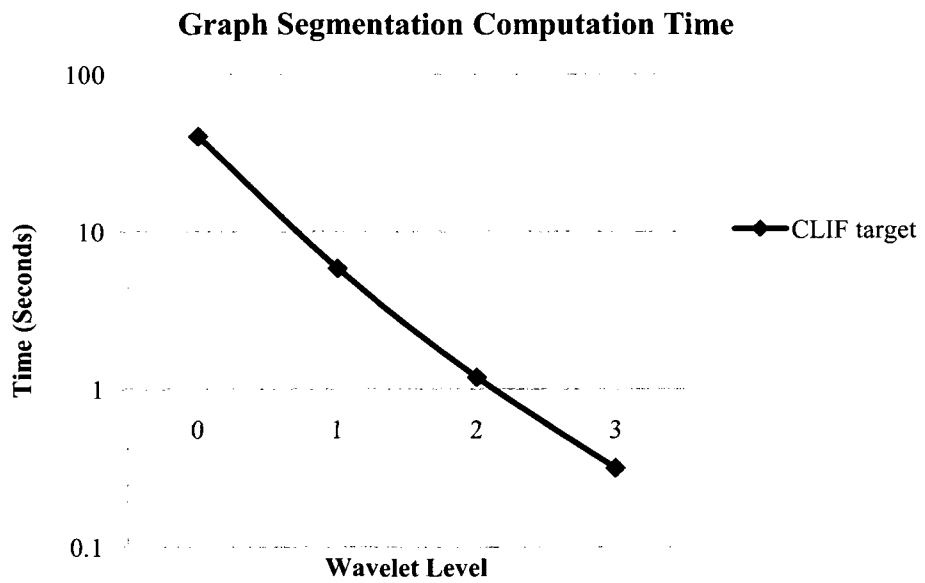


Figure 4.9 Graph Segmentation of CLIF Objects Computation Time

4.3.7 Conclusion

Although object boundaries are coarsely approximated at a wavelet level of 3, time and accuracy have shown to be optimal at this level. The rectangular shape of the object lends itself to coarse approximations derived from wavelet decomposition. In the future, study graph segmentation implementations on more datasets will be studied and detection results with other segmentation algorithms will be compared, as well as test new types of higher-level statistics to improve segmentation.

4.4 Data-Mining-Based Moving Object Detection

Taking concepts from the previous two methods, a comprehensive moving object detection scheme that uses object prior offline machine learning to detect objects in an aerial image sequence autonomously has been developed. Object detection and tracking in low frame rate, low resolution video poses challenges that are difficult for recent object tracking methods to accurately detect and track objects. In this section, a multi-stage algorithm which takes image sequences and object priors as input to first train an offline classifier is presented which then uses similar real time data to detect moving objects. After preprocessing and optical flow estimations are completed, motion, intensity, and edge features are used to create multiscale graph segmentation of the moving image sub-field. Object candidates are found after post processing using object prior information, where a second set of features are calculated and input into the trained classifier.

The primary contribution is to introduce novel feature sets specifically designed for moving object segmentation in graph theoretic and object silhouette detection via binary supervised classification. The novel set of edge histogram features are generated from

each candidate segment (no background information used) and fed into a binary classifier that is trained offline with a set of ground truth data, achieving real-time results. The final stage of the algorithm uses detected candidates between subsequent frames to suppress objects with weak size and spatial similarity, allowing only object candidates with highest similarity to a previous candidate remain. Each step of this algorithm will be detailed, referencing previously described methods.

4.4.1 Motion Detection

This method implements the phase-based optical flow with wavelets flow described in Section 4.2.3, using level 2 wavelet approximation as input to get optical flow output. Optical flow values, O , are then post processed to reduce non-vehicular motion due to background noise and camera movement. Both a size and magnitude filter are used. The filters are based on object dimensions to reduce motion detection to areas a $\frac{1}{4}$ minimum object dimension, d_{min} , (for cases of partial occlusion) to 4 x maximum object dimension, d_{max} , (for cases where multiple objects are traveling together). Minimum size, $\tau_{minsize}$, and magnitude, τ_{minmag} , thresholds are used to reduce noise, while camera motion is detected and rejected by finding a dominant flow direction, $\tau_{domorient}$ that indicates camera motion.

After each of the post processing steps are applied, a reduced set of locations, G , for frame t where potential object motion is present, and can be written as:

$$G_t = \left\{ s \in (\Omega_j) \mid O_{s,\{u,v\}} > \tau_{minmag} \ \& \ N_s > \tau_{minsize} \ \& \ \tan\left(\frac{O_{s,v}}{O_{s,u}}\right) \neq \tau_{domorient} \right. \quad (4.28)$$

where $O_{s,\{u,v\}}$ is the optical flow in the u and v at s , and N_s is the 4-connected neighborhood around s with $O_{s,\{u,v\}} > \tau_{minmag}$. The orientation of $O_{s,\{u,v\}}$ is collected

into discrete bins and first used to find a possible dominant orientation, then used for object motion location determination.

4.4.2 Flow Gradient Intersect Features

Due to the constraints necessary to find optical flow, aperture problem errors can cause inaccurate velocities. Most errors include motion detected along the direction of intensity gradients, which occur along object boundaries. Moving objects typically display a concave boundary, which causes an intersection near the object centroid when line segments are extended from points of detected optical flow along its orientation. The line segment length, r_{flow} , is chosen to extend half the distance of the maximum object dimension from the origin of the optical flow value.

Each line segment is then overlaid onto a 2-D grid, I_f , of same dimensions as the optical flow input, where the total sum of intersecting lines is tabulated at each location. A Gaussian filter is then applied to the grid to smooth the values, causing peaks to appear near object centroids (Figure 4.10).

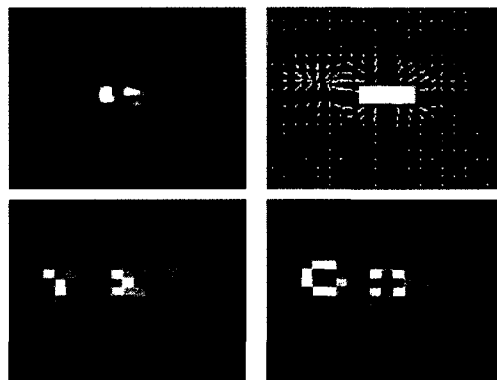


Figure 4.10 (*Top left*) The original image and (*top right*) its 2-Level 2-D wavelet approximation coefficients (*bottom left*) overlaid with optical flow field, the corresponding flow gradient image is shown, and (*bottom right*) the final image after a 2-D Gaussian filter is applied.

These types of clusters of high values in the flow gradient images are very common in both rotational and translational motions of objects. In Figure 4.10, the left cluster corresponds to the white object's location in the second frame input in the optical flow image.

4.4.3 Edge Intersect Features

As seen in Figure 4.10, two clusters of flow gradient intersect values correspond to locations of the same object in the two optical flow input frames. An additional feature set that will distinguish between the object's location in both frames is provided. The concave boundary of objects is used to find locations where line segments normal to object edgelets (small edge segments) intersect at high consistencies.

A procedure similar to that described in Section 4.4.2 is used to determine gradient orientations of boundaries. First, a Canny filter is used to create an edge image of the optical flow input images. After edge images have been calculated, windowed regions of size w_{edge} are analyzed for dominant edgelet orientation. For the implementation, a $w_{edge} = 3 \times 3$ window is used, giving possible orientations of 0° , 45° , 90° , 135° . A line segment of length r_{edge} is created at the edge point in the direction normal to the dominant edgelet orientation. A 2-D accumulator grid, I_e , is then generated and smoothed in similar fashion as described in Section 4.3.2. Figure 4.11 gives typical results of the edge flow features for an object.

This feature set will be implemented in both the graph segmentation, object candidate classification, and non-maximal suppression stages of the algorithm. The specific methods for implementation will be discussed in Sections 4.4.4-4.4.8.

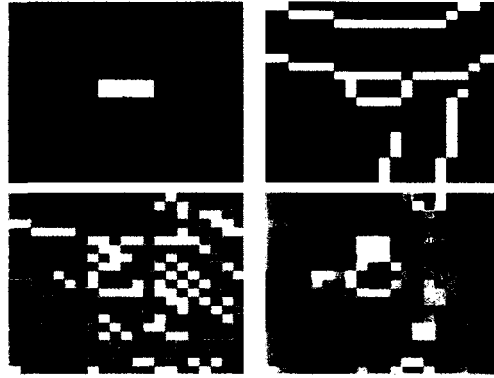


Figure 4.11 Edge Gradient Flow Process

4.4.4 Hierarchical Graph Segmentation

In this next stage, an initial graph, $G_o^{(t)}$, is created from G_t , where the initial nodes, $V_o^{(t)} = s^{(t)}$, assuming that members of $s > 0$ (candidate object motion is detected).

Initial undirected edges, $E_o^{(t)}$, are then formed between nodes spatially located within a predetermined distance, d_{edge} . The initial edge energy cues are comprised of a feature vector, x , which includes values from the wavelet approximation, flow gradient intersect, edge gradient intersect, and location. The feature set of $G_o^{(t)}$ can be written as for each ($i = 1 \dots |G_t|$):

$$x^{(i)} = (x_1^{(i)}, x_2^{(i)}) \quad (4.29)$$

where

$$x_1^{(i)} = (x, y), \quad x_2^{(i)} = [I_{w,j}(x, y), I_f(x, y), I_e(x, y)].$$

A weight matrix is created, $W_o^{(t)}$, and populate it with the edge weights between neighboring nodes, using the following energy function taken from [44]:

$$w_{ik} = \begin{cases} e^{-\alpha |x_2^{(i)} - x_2^{(k)}|} & \text{if } 0 < \|x_1^{(i)} - x_1^{(k)}\| < d_{edge} \\ 0 & \text{otherwise} \end{cases} \quad (4.30)$$

where α is a vector of weights corresponding to each of the members in $x_2^{(i)}$ and i and k are members of node set $V_o^{(t)}$. Sharon [41] concluded that when solving the generalized eigen problem $Lu = \lambda Wu$ with minimal positive eigenvalue λ , salient nodes are found where $\Gamma(u)$ is minimal (as described in Section 4.3.2). In this formula, u is the set of membership nodes, which is initially set as:

$$U_o^{(t)}(i, k) = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases} \text{ for } i, k = 1, 2, \dots |G_t|. \quad (4.31)$$

The median value of Γ is chosen as the threshold in determining which nodes are selected as node locations at the next coarsest level. Setting $1 > \beta > .5$ will increase the tendency for larger nodes to be chosen as new coarse nodes, where $.5 > \beta > 0$ will cause smaller node selection to increase. $\beta = .5$ eliminates any size bias in the node selection. Depending on the dimensions of the image frame compared with the desired object dimension, β can be automatically selected to create nodes of a size similar to prior object dimensions.

Higher scales of nodes are then determined using seed nodes with low Γ values, with interpolation matrices used to associate different scale node members with one another. Mean node feature values are determined at higher scales based on the original $x_2^{(i)}$ features, as explained in Section (4.3.3). The hierarchical node agglomeration stage halts when the number of nodes size remains constant.

Figure 4.12 demonstrates the hierarchical graph segmentation process, which achieves strong segmentation of object regions through combining similar nodes having feature values described in Sections 4.4.2-4.4.3.

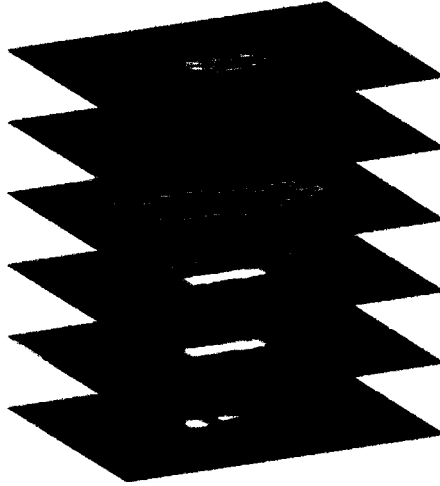


Figure 4.12 Hierarchical Node Agglomeration Using Multi Cue

4.4.5 Object Candidate Selection

After G^t and G^{t+1} (the two frames associated with optical flow) have been calculated, the next step is to iterate through each level of graph nodes and collect nodes that resemble object prior spatial statistics into an object list, $T^{t,t+1}$. The graph node area is used as minimum and maximum dimensions as spatial features, which are compared to object priors for inclusion into $T^{t,t+1}$. Object candidate membership is formulated as:

$$T^{t,t+1} = G_l^{(t)}, G_k^{(t+1)} \left\{ \left| \tau_{maxarea}, \tau_{maxdim} > V_{li}^{(t)}, V_{kj}^{(t+1)} > \tau_{minarea}, \tau_{mindim} \right. \right. \quad (4.32)$$

$$\text{for } l = 0 \dots |G^t|, k = 0 \dots |G^{t+1}|,$$

$$i = 1 \dots |G_l^{(t)}|, j = 1 \dots |G_k^{(t+1)}|$$

All object candidates are passed into the object detection through binary classification stage described in Section 4.4.6.

4.4.6 Supervised Classification Object Detection

Rotationally invariant feature sets are found using the centroid of each object candidate in $T^{t,t+1}$. Due to high variance in the possibly present object intensity, only the edge feature data is used to determine the probability of object presence. The first set of rotationally invariant features are a normalized set of the sum of edges binned in a set of radial distances from the centroid, termed edge histograms (explained in full detail in Chapter 5). This feature set, p_r , can be treated as a probability density function (pdf), written as:

$$p_r(r_k) = \frac{n_k}{n} \quad k = 0,1,2, \dots, L - 1, \quad (4.33)$$

where r_k is the edge pixel radius bin from the centroid, and n_k is the total occurrences of edge pixels within radius distance bin k , and n is the total number of edge pixels found.

A second set of edge features uses the edge gradient intersect features associated with each object candidate centroid in $T^{t,t+1}$. The sum of intersect values occurring in the same radial distances used for the edge histograms is now taken to normalize the values to create a pdf, p_g , of gradient intersects occurrences, with each value written as:

$$p_g(r_k) = \frac{b_k}{b} \quad k = 0,1,2, \dots, L - 1 \quad (4.34)$$

where $b_k = \sum I_e(x_i, y_i), \quad i \in r_k$.

Due to low signal to noise ratio (SNR) at the higher wavelet levels necessary to achieve real time optical flow results, the known up sampling index correspondences of

the centroid at $j - 1, j - 2, \dots, 0$ is also used. Sets of features at each of these wavelet levels are then obtained, as well, giving us a final feature set:

$$f_i = (p_r^j, p_r^{j-1}, \dots, p_r^0, p_g^j, p_g^{j-1}, \dots, p_g^0), \quad (4.35)$$

$$i = 1, 2, \dots, |T^{t,t+1}|.$$

Now that object candidate features have been calculated, each object feature vector is used as input into a binary classifier trained offline using ground truth object data. Offline training uses object ground truth centroids and non-object locations that generate optical flow due to object position changes, thus allowing for fast binary classification of object candidates.

4.4.7 Multi-Frame Object Matching

Now that most non-object candidates have been eliminated using the method described in Section 4.4.6, objects, which are detected in sequential frames, must be retained. An object matching through Mahalanobis distance measure, which first iterates through remaining objects in $T^{t,t+1}$, deemed $T_{targ}^{t,t+1}$, is used to find the best matches between members of T_{targ}^t and T_{targ}^{t+1} that occur within maximum object translation, d_{trans} . The following previously calculated histogram features p_r^j and p_g^j are used, as well as introduce a three dimensional histogram of optical flow values:

$$p_{of}(|o_{ijk}|) = \frac{f_{ijk}}{f} \quad i = 0, 1, 2, \dots, L_{orient} - 1, \quad (4.36)$$

$$j = 0, 1, 2, \dots, L_{mag} - 1, k = 0, 1, 2, \dots, L_{dist} - 1,$$

where o_{ijk} is a discretized optical flow value occurring r_k from the centroid, with the optical flow having the i th orientation and j th magnitude. The value f_{ijk} is the total

number of occurrences of the absolute values that occur in bin ijk , and f is the total non-zero optical flow values. Absolute values of the flow are used to ensure compliment gradient direction will be treated is strong indication of similar flow.

$$Mdist_i^t = \underset{j}{\operatorname{argmin}} \left[\frac{\operatorname{mahal}(\mathbf{p}_r^{(i,j)}, \mathbf{p}_r^{(k,j)}) + \operatorname{mahal}(\mathbf{p}_g^{(i,j)}, \mathbf{p}_g^{(k,j)}) + \operatorname{mahal}(\mathbf{p}_{of}^{(i,j)}, \mathbf{p}_{of}^{(k,j)})}{3} \right] \quad (4.37)$$

$$\text{if } \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2} < d_{trans},$$

$$i = 1, 2, \dots, |T_{targ}^t|, k = 1, 2, \dots, |T_{targ}^{t+1}|.$$

Values in T_{targ}^t that have no matches or matches above a predefined threshold are removed from the object candidate list, reducing false alarms that resemble objects due to clutter, shadows, and illumination effects.

4.4.8 Object Non-Maximal Suppression

Multiple candidates represent segments of the object at different node scales, which can lead to multiple successful object matches in 4.4.6 of the same object. It is necessary to determine which of these candidates best represents the object regions, thus, T_{targ}^t suppresses objects within a minimum distance, d_{min} , of each other, leaving only the best object unsuppressed. The suppression decision is based on object candidate areas closest to optimal object prior area, a_{prior} , which is written as:

$$T_i^t = \begin{cases} targ & \text{if } \underset{j}{\operatorname{argmin}} |a_{prior} - a_j| \text{ for all } i < d_{min} \\ nontarg & \text{otherwise} \end{cases} \quad (4.38)$$

$$i = 1, 2, \dots, |T_{targ}^t|$$

This necessary step significantly reduces the number of multiple detections of the same object, but should be performed after the supervised classification to ensure the non-objects similar to object areas do not suppress actual object segments. Figure 4.13 gives

an example of how the object detection and non-maximal suppression reduce object false alarms.

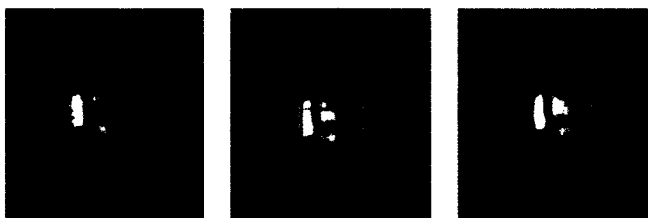


Figure 4.13 Binary Classification and Non-Maximal Suppression Results of Object Candidates

4.4.8 Object Detection Testing Data

A subset of the camera 1 Columbus Large Image Format (CLIF) sequence dataset is chosen for testing (described in Section 4.2.4). The object ground truths is increased to 39 objects (Figure 4.14) with a total of 322 potential object detections. Non-object ground truth were chosen as locations outside object ground truths that displayed optical flow values in the range to pass through size and magnitude filters.



Figure 4.14 Object Detections in CLIF Data Sequences

4.4.9 Object Detection Evaluation

For evaluating the object detection framework, the accuracy of the algorithm is determined by comparing segmentations labeled as “objects” against segmentation

ground truth of moving objects. A detection is considered “positive” if the centroid of the algorithm output and the object ground truth are within a minimum distance threshold, set to 5 pixels for the data set. False negatives are accumulated when moving objects go undetected after appearing for the necessary minimum frames (two in the case). The results are presented using differing levels of object candidate matching thresholds, which allow higher accuracy with more false alarms as the matching distance is relaxed, but allows for more false positive results (Figure 4.15).

To ensure selection of a classifier with the highest classification accuracy and speed, the feature set has been tested on several machine learning methods, with their evaluation discussed in the subsequent sections. As can be seen in Figure 4.16, Random Forest and Association Rule (PART) classifiers outperforms other methods with a 98.6% and 98.2% area under the curve (AUC) of the receiver operator characteristic (ROC), respectively.

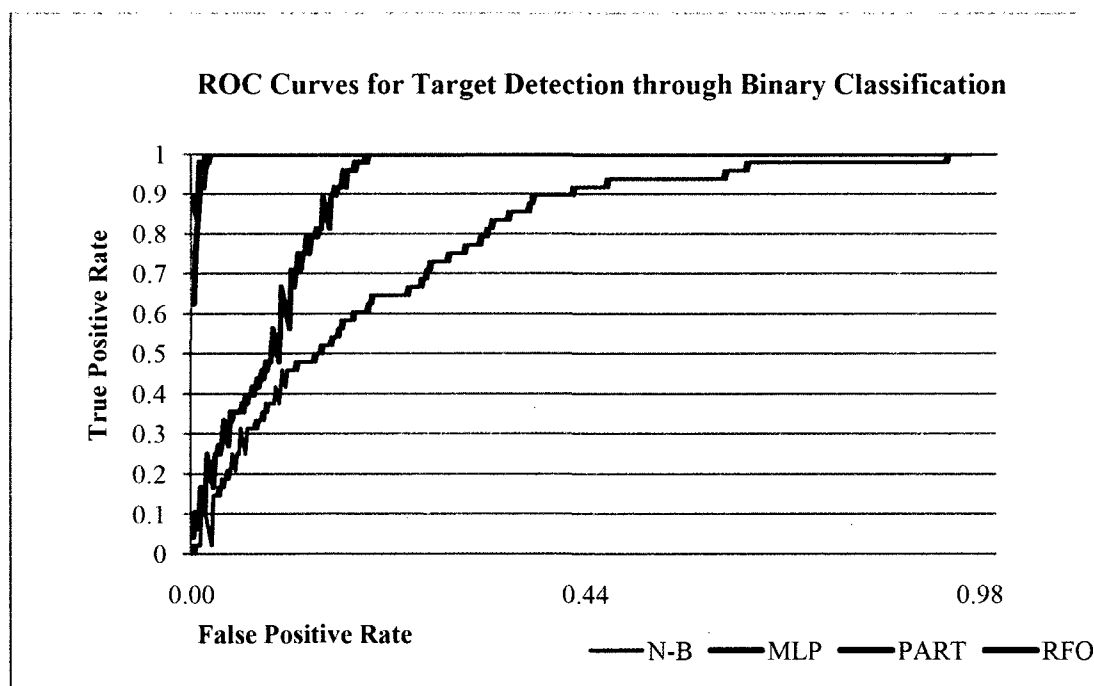


Figure 4.15 ROC Curves of Binary Object Classification

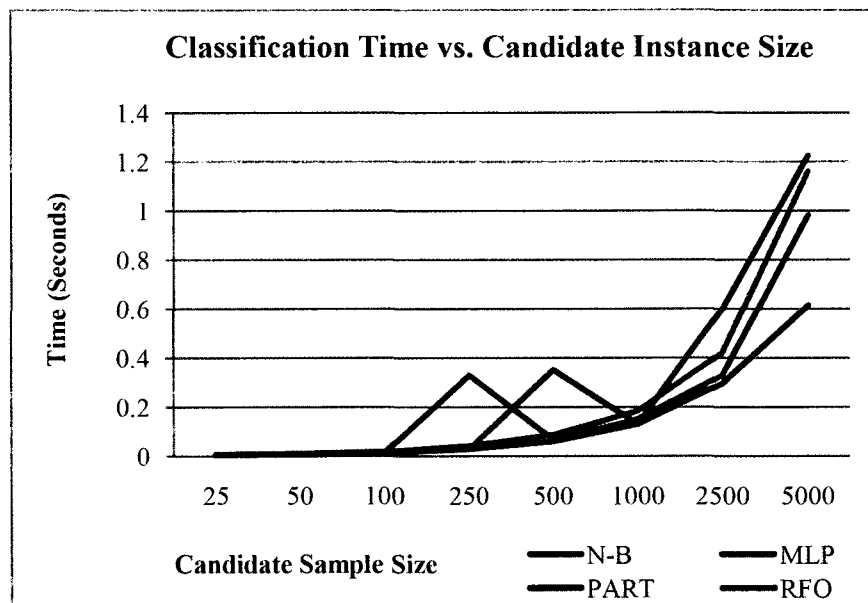


Figure 4.16 Classifier Execution Speeds for Varying Object Candidate Sizes

4.4.10 Conclusion

The combination of optical flow, edge and wavelet-based feature sets can effectively differentiate objects from the background in low frame rate videos for object detection in real time. The framework and stages necessary have been used to achieve high detection accuracy by using phase-based optical flow and hierarchical graph segmentation, coupled with an offline-trained supervised classification method for object discrimination.

Intermediate filtering stages reduce false alarms and poor segments, along with multiple detections through object matching and non-maximal suppression. Further work includes expanding the method into tracking objects using multiple object detections coupled with a dynamic state model.

4.5 Object Detection Conclusion

Several object detection methods that can be implemented on a variety of sensor data that has clearly defined objects have been presented. Properties such as multiscale wavelet transformations and hierarchical graph segmentation allows for the user to adjust the segmentation and motion estimation outputs to perform optimally for a given object library shape and size. It is the belief that algorithm performance varies for each detection method under specific scenarios, where a multi-detection framework (such as the one described in Section 4.4) provides optimal performance over the range of OCs that occur in SE.

CHAPTER 5

FEATURE EXTRACTION AND CLASSIFICATION

5.1 Feature Extraction and Classification Introduction

As discussed in Chapter 3, selecting object characteristics that robustly discriminate themselves from cluttered backgrounds is imperative for successful classification accuracy. In this chapter, feature sets and classifiers that achieve these specific aims within the SE framework (Figure 5.1) are discussed.

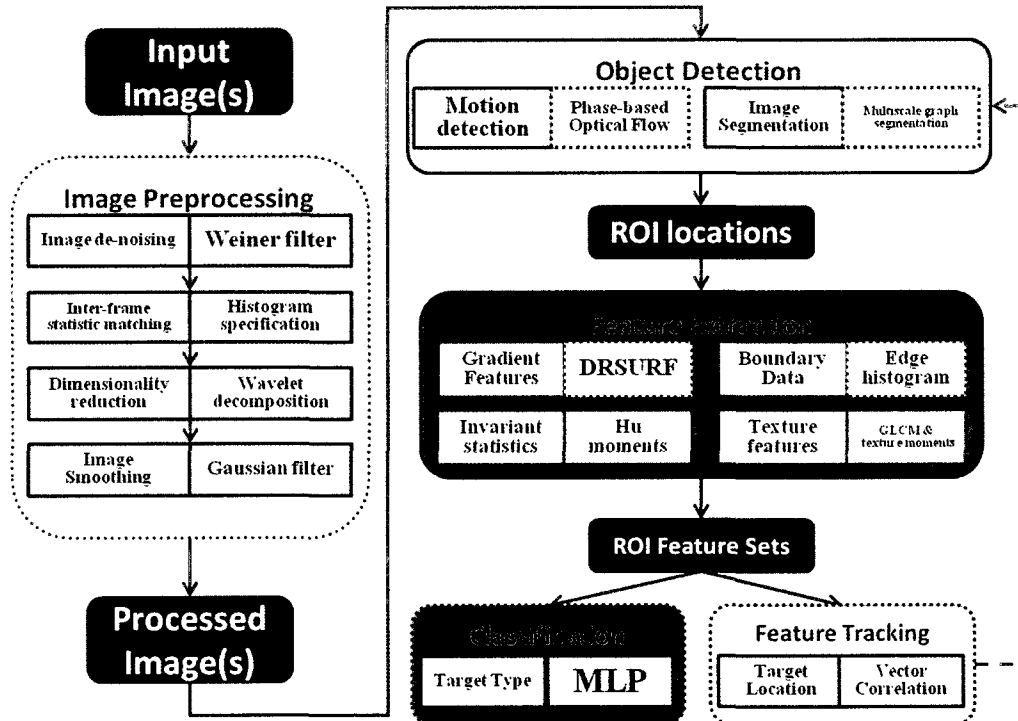


Figure 5.1 Feature Extraction and Classification in the SE Framework

First, a novel set of features, dense and reduced speeded-up robust features (DRSURF), which are used for tracking (described in Chapter 6), are introduced. Next, feature sets on both binary classification (detection) and multi-class classification on SE data sets are presented and evaluated. These feature sets include another novel feature set, edge histograms. The chapter will end with a conclusion on the feature sets discussed, with recommendations on how and when to implement them for other SE data sets.

5.2 Dense and Reduced Speed-up Robust Features (DRSURF)

The dense and reduced SURF (DRSURF) descriptors, as the name implies, are a dense set of SURF descriptors using a smaller area and gradient bin size. If an interest point detection methods using a Hessian determinant or any other corner point detector does not find repeatable locations to calculate descriptors for low resolution objects, this issue is combated by taking reduced scale gradient values at pixel locations evenly spaced within the size of the objects of interest. Issues of orientation variations are discussed in this section as well.

5.2.1 DRSURF Interest Point Detection

An integral image is used to quickly calculate the area of rectangular regions (the operations necessary). Due to an approximately constant scale of objects in the image sequence for aerial object tracking, the scale space feature calculations that SIFT and SURF both employ are disregarded, although an extension to include multiscale descriptors is easily possible. The method deviates from SURF in the choice of scale space interest points; instead motion detection from the optical flow step in the algorithm

to find the points of interest is used. Areas in the optical flow field that have motions large enough to be a moving object are chosen as interest points. A grid is then created on top of the interest point that spans p_d pixels in every direction from the central point. The value of p_d is chosen empirically to encompass the entire potential object area through adjusting the object point window size w_p , with total window size being $m_p = 2w_p \times 2w_p$. A step variable, $s_{Gridstep}$, is also used to determine the spacing between interest points, along with p_d , yields the density of the feature vector. After testing different combinations of p_d and $s_{Gridstep}$, it was determined that $p_d = 3$ and $s_{Gridstep} = 2$ provides the best discrimination between object and background for the entire data set, but $p_d = 2$ and $s_{Gridstep} = 2$ provides reasonably high results for the reduced computation (Figure 5.2).

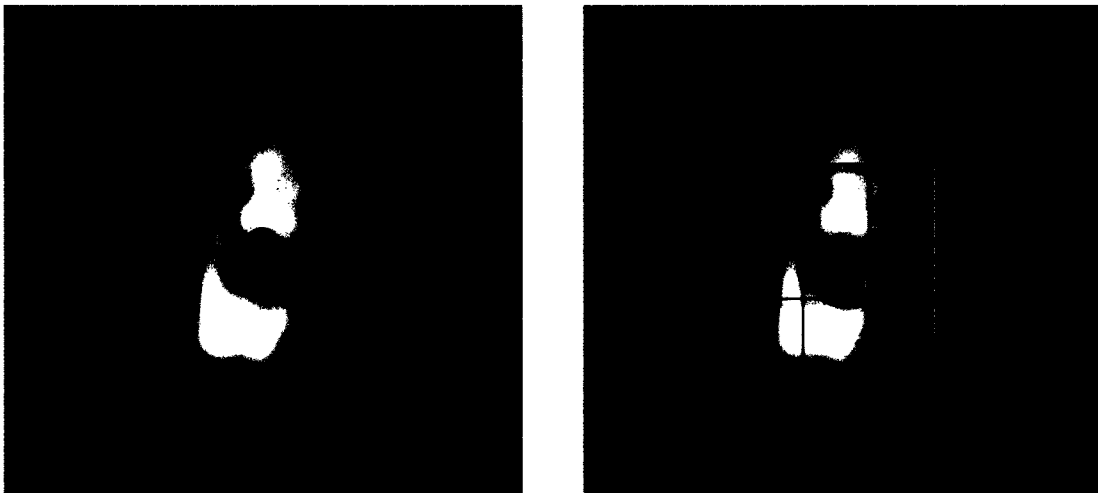


Figure 5.2 DRSURF Interest Point Location and Feature Window

5.2.2 DRSURF Orientation Assignment

Reproducible orientation methods that are employed in SIFT and SURF to determine the orientation around an interest point that yields a maximum gradient magnitude. As the image rotates, this gradient magnitude will change, along with the orientation, which ensures the correct feature descriptor alignment. Because the interest points typically lack a large gradient, a set of feature descriptors with multiple orientations is calculated by rotating the interest point grid by $\pm\theta * s_{Anglestep}$. The feature descriptors are determined by prior knowledge of object behavior and gradient invariability. In the case, objects cannot rotate more than $30^\circ/frame$, and feature descriptors maintain rotational invariability to 15° , so $\theta = 15^\circ$ and $s_{Anglestep} = 2$ is used, giving five orientations per interest point (Figure 5.3 shows a subset of these). This increase in dimensionality and computation is necessary and possible due to the reduced set of interest points from the optical flow calculation and reduced size of the gradient descriptor.

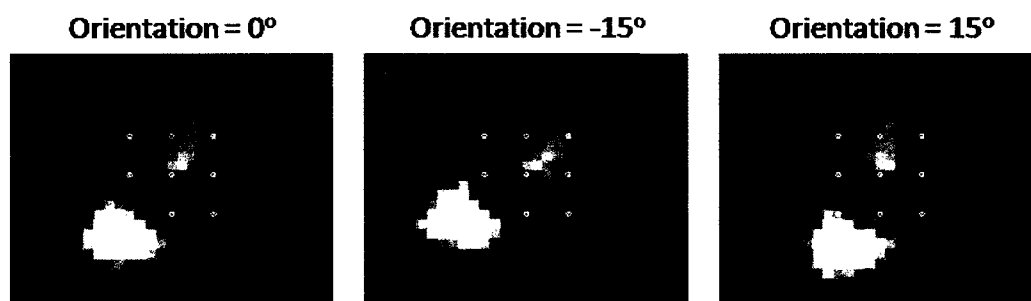


Figure 5.3 DRSURF Grid Over Multiple Orientation

5.2.3 DRSURF Feature Descriptor Calculation

To explain the method of calculating gradient bins within an object area, the original SURF method, which is based on the SIFT feature descriptor [45], is discussed. SURF

uses approximations to create descriptors similar to SIFT but with reduced computations while sacrificing very little matching accuracy [46]. The first step in SURF and SIFT is to find interest points by identifying locations with large gradients in both the x and y directions (corner points), which is not implemented in the next step. Following the SURF method, the integral image of the input image I_1 is first used to speed up computation of the gradients in the descriptors step. This step can be formalized as:

$$I_{\Sigma}(x, y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(x, y) . \quad (5.1)$$

The next step is to use the interest point grid discussed in Section 5.2.1 to construct square windows around each of the points on the grid. The window around each point is 18 x 18 pixels in size. This size is based on the scale chosen to extract gradient bins. The descriptor window is divided into 2 x 2 regular subregions. Within each of these subregions, the Haar wavelets of size 6 are calculated at the locations within the specific subregion. The Haar Wavelet responses for each subregion find the gradients in the x and y direction spanning a scale (six in the case) by using box filters and the integral images, which requires only six calculations. The vector for each bin describes the underlying intensity structure of the neighborhood as $v = [\sum d_x \quad \sum d_y \quad |\sum d_x| \quad |\sum d_y|]$. The gradient for each point in the interest point grid then becomes a set of descriptor vectors of size 4 x 4. Combining the entire interest point grid creates an interest point vector of size $4 \times 4 \times \left[2 * \text{floor} \left(\frac{p_d}{s_{Gridstep}} \right) + 1 \right]^2$. Consequently, for $p_d = 2$ and $s_{Anglestep} = 2$, an object can be represented with a 144 member length descriptor vector.

5.3 Invariant Moment Descriptors

As briefly mentioned in the Chapter 2, invariant moment descriptors provide a set of statistical features that have considerable invariance to rotation, translation, and scaling changes in object images [81]. This set of descriptors combines the overall shape and intensity distribution, which can be formulated using the discrete version of the moment for an image $I(x, y)$:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y), \quad (5.2)$$

to then be able to find centroids $\bar{x} = \frac{M_{10}}{M_{00}}$, $\bar{y} = \frac{M_{01}}{M_{00}}$. The *central moments* are then defined as:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I(x, y), \quad (5.3)$$

where p and q give the moment order. The central moments used in the Hu moments are written as:

$$\mu_{00} = M_{00}, \quad (5.4)$$

$$\mu_{01} = 0$$

$$\mu_{10} = 0,$$

$$\mu_{11} = M_{11} - \bar{x}M_{01} = M_{11} - \bar{y}M_{10},$$

$$\mu_{20} = M_{20} - \bar{x}M_{10},$$

$$\mu_{02} = M_{02} - \bar{y}M_{01},$$

$$\mu_{21} = M_{21} - 2\bar{x}M_{11} - \bar{y}M_{20} + 2\bar{x}^2M_{01},$$

$$\mu_{12} = M_{12} - 2\bar{y}M_{11} - \bar{x}M_{02} + 2\bar{y}^2M_{10},$$

$$\mu_{30} = M_{30} - 3\bar{x}M_{20} + 2\bar{x}^2M_{10},$$

$$\mu_{03} = M_{03} - 3\bar{y}M_{02} + 2\bar{y}^2M_{01}.$$

From these, moments η_{ij} , can be made invariant to translation and scale by dividing by the scaled 00th moment, written as:

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00} \left(1 + \frac{i+j}{2}\right)}. \quad (5.5)$$

Finally, the seven Hu moments, φ_i

$$\varphi_1 = \eta_{20} + \eta_{02}, \quad (5.6)$$

$$\varphi_2 = (\eta_{20} + \eta_{02})^2 + (2\eta_{11})^2,$$

$$\varphi_3 = (\eta_{30} + 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2,$$

$$\varphi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2,$$

$$\begin{aligned} \varphi_5 = & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - \\ & 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - 3\eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \\ & \eta_{12})^2 - (\eta_{21} + \eta_{03})^2], \end{aligned}$$

$$\begin{aligned} \varphi_6 = & (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + \\ & 4\eta_{11}(\eta_{30} - \eta_{12})(\eta_{21} - \eta_{03}), \end{aligned}$$

$$\begin{aligned} \varphi_7 = & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - \\ & 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \\ & \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]. \end{aligned}$$

As noted in Chapter 2, invariant moments have been successfully implemented for feature tracking. This success is the main justification for evaluating these features for multi-class object classification. This feature set is also useful because of its ability to both reduce an image to seven values and to provide good discrimination even with multiple variations in image appearance (Figure 5.4). Invariant moments were calculated

at each wavelet approximation level for every object. Classification discrimination is provided in the evaluation and conclusion sections.

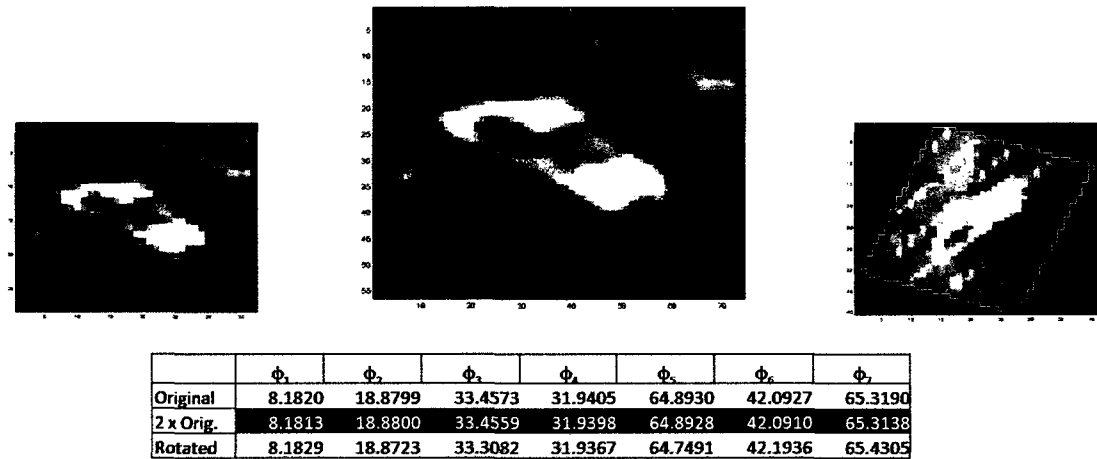


Figure 5.4 Scaled and Rotated Images and Corresponding Invariant Moments

5.4 Edge Histograms

To provide more discrimination of object shape, a method that combines edge detection and histogram analysis is employed to find a rotational invariant set of descriptors, which are deemed *edge histograms*. Because this method is not scale invariant, wavelet approximations are resized to the original image dimensions using the cubic interpolation.

To find edge histograms, first run the *canny edge filter* on the rescaled wavelet approximations (Figure 5.5).

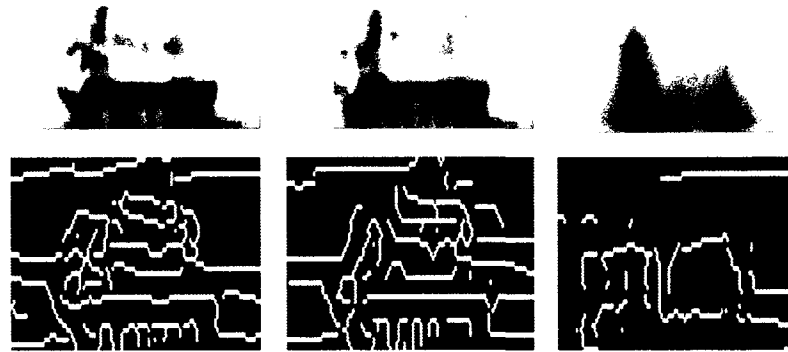


Figure 5.5 Canny Filtered Objects at Varying Wavelet Levels

After the binary edge image is calculated, the center of the image is chosen as the origin and circular bins of five pixel radii intervals are created, summing the total amount of detected edge pixels occurring within the specified range (Figure 5.6). This largest bin is chosen to be $r = 50$, which is equal to half of the length of the largest object in the dataset. The bin vector is itself used as a feature vector for classification for both binary classification and multi-object type classification, with evaluation results provided in Section 5.5.

Edge histogram statistics are also calculated to find a more robust statistical representation of the edge distribution. The *mean*, *standard deviation*, *skewness*, and *kurtosis* (described in Section 5.5) are chosen.

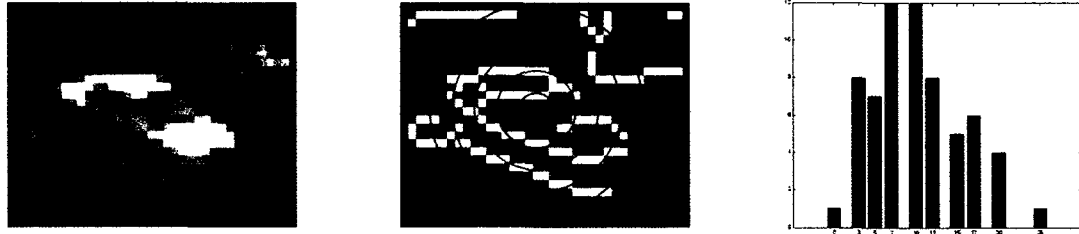


Figure 5.6 Edge Bin Histogram of an Object Image

5.5 Texture Statistics

A set of texture statistics is calculated directly using the wavelet approximation to gain insight on how different levels of object detail effects the specificity and sensitivity of object texture features for classification. The intensity histogram is then used to account for rotational variations in the image data sets. Here, a set of features that is based on the coefficient intensity histogram is described, which is written as a probability density function (pdf):

$$p_r(r_k) = \frac{n_k}{n} \quad k = 0, 1, 2, \dots, L - 1, \quad (5.7)$$

where $p_r(r_k)$ is the probability of an image pixel being a member of histogram bin r_k .

For this method, $k = 16$ bins is chosen for the texture calculations, unless noted otherwise. To calculate the moments, the moment equation is written as follows:

$$\mu_n(r) = \sum_{k=0}^{L-1} (r_k - m)^n p(r_k), \quad (5.8)$$

where the mean value of r is:

$$m = \sum_{k=0}^{L-1} r_k * p(r_k). \quad (5.9)$$

The second, third, and fourth moments, which are the *standard deviation*, *skewness*, and *kurtosis*, respectively, are calculated. Also, the *uniformity* is calculated as:

$$U = \sum_{k=0}^{L-1} p^2(r_k) \quad (5.10)$$

and the *entropy* to measure variability as:

$$e = -\sum_{k=0}^{L-1} p(r_k) \log_2 p(r_k). \quad (5.11)$$

Besides calculating the statistical moments of the gradient histogram, the gray-level co-occurrence matrix is found for each of the principal directions ($0^\circ, 45^\circ, 90^\circ, 135^\circ$) to find how often pixels of similar intensity “co-occur” within an image in the orientation specified, that will produce a square matrix, G , the size of intensity bins, k , in each dimension. The value at each location $G_{i,j}$ will be the total number of times two pixels of a chosen orientation having similar intensity values “occur” within the region. After constructing co-occurrence matrices in the principal directions (typically $0^\circ, 45^\circ, 90, 135^\circ$), the matrices are normalized to give a joint occurrence probability of pixel pairs with the corresponding orientation and intensity range (Figure 5.7).

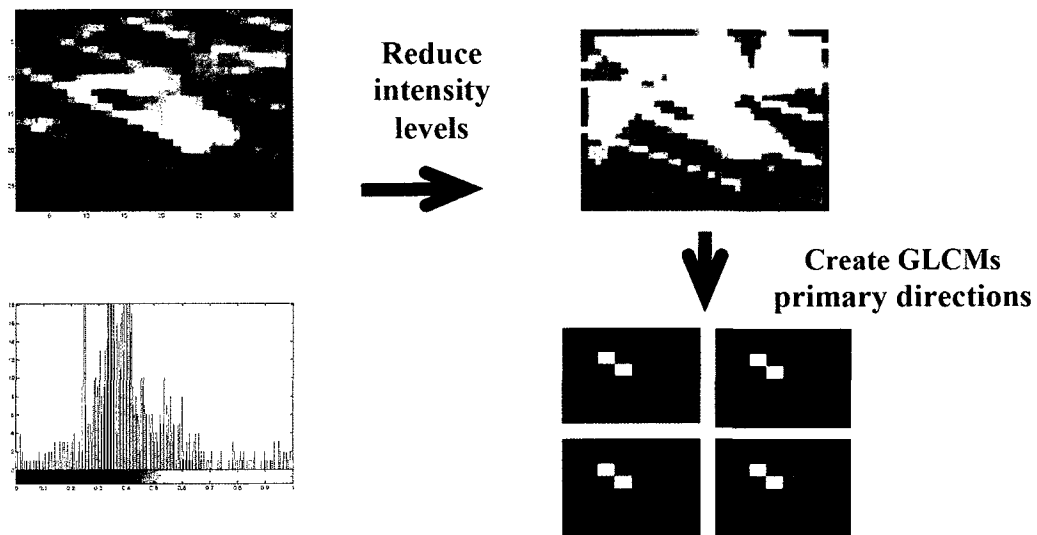


Figure 5.7 Steps in Texture Calculation

A set of spatially dependent texture descriptors termed Haralick features [56] are defined as:

1. Contrast: $\sum_{i,j} |i - j|^2 p(i, j)$
2. Correlation: $\sum_{i,j} \frac{(i - \mu_i)(j - \mu_j) \bar{p}(i, j)}{\sigma_i \sigma_j}$
3. Energy: $\sum_{i,j} p(i, j)^2$
4. Homogeneity: $\sum_{i,j} \frac{p(i, j)}{1 + |i - j|}$

Contrast measures intensity differences between neighboring pixels over the entire image, where *correlation* returns a measure of how correlated a pixel is to its neighbor. The *energy* provides a sum of the squared elements in the co-occurrence matrix, returning a value of 1 for a constant intensity. *Homogeneity* provides a value of the closeness of the member distribution in the co-occurrence matrix. To address rotational invariance, an average of the feature values is taken for all the principal orientations, so that co-occurrences that move to a different matrix due to rotational changes will still be measurable.

5.6 Classification Methods Using Feature Descriptors

Classification methods that have been previously used to classify objects both for detection and object type schemes have been found. These methods will be discussed further in Section 5.6.1.

5.6.1 Bayesian Classification

Bayesian classification leverages prior probabilities from observed evidence to determine whether a specific hypothesis is true or false. This method is powerful when combined with domain expert knowledge, given that a probability can be numerically

evaluated. *Naïve Bayes* classification can use maximum likelihood estimates of probabilities based on distributions of the data, allowing for classification of states without domain knowledge. This method decouples conditional feature distributions, so that fused data sources can be independently estimated as a one-dimensional distribution. This method, too, can be used in a hierarchical manner, which allows for feed-forward inputs of lower-level fusion classification results into high level of classification.

5.6.2 Neural Networks

This machine learning method allows classifiers to be continuously trained through back propagation and supervised learning to increase the accuracy of classification, modeling the synaptic processes of neurons. The method can leverage domain expertise to classify threats and conditions, which can then be used to retrain the network through back propagation. Two types of neural network classifiers are implemented for evaluation, multilayer perceptron and radial basis function methods. *Multilayer Perceptron Learning (MLP)* uses multilayer perceptron, adaptive learning, and back propagation to increase the accuracy of classifiers. *Radial Basis Function (RBF) network* typically has three layers: an input layer, a hidden layer with a non-linear RBF and activation function, and a linear output layer.

5.6.3 Support Vector Machines

Support Vector Machines (SVM) are a set of related learning methods used for classification and regression analysis. In SVM, a hyperplane or set of hyperplanes is constructed. SVMs work best with data sets that have a wide “functional margin.” *Sequential Minimal Optimization (SMO)* trains a support vector and solves (classifies)

data analytically by breaking down the size of the dataset. SVM has empirically been shown to give a good generalization performance on a wide variety of problems.

5.6.4 Association Rule Mining

Association Rule Mining is a market basket approach that finds connections between the occurrences of different features occurring within a class with a level of *support* and *confidence*. A rules-based method known as PART algorithm that obtains rules from partial decision trees is obtained and then used to build a tree using the decision tree learner [82].

5.6.5 KSTAR Classification

KSTAR is an instance-based classifier that bases the test instances upon the class of training instances similar to it. In KSTAR, the similar class training instances are determined by an entropy distance measure. This method provides a consistent approach to handling attributes of different types, as is the case in the evaluation [83].

5.7 Feature Descriptor and Classification Evaluation

For testing classification accuracy for the feature sets, three datasets that provide different challenges for successful classification have been employed. The datasets have been approved for public release, and all ground truth data was created through visual inspection by the authors. All classification was performed using Weka 3.6.2 machine learning software and feature descriptor calculations were computed using Matlab R2009a on an Intel® Core i7 920 @ 2.67 GHz processor with 12 GB of RAM.

The feature descriptor sets have been tested at wavelet levels $N = 0,1,2,3,4$ for CLIF dataset and $N = 0,1,2,3,4,5$ for CEGR and IICO with varying degrees of accuracy. We will present and discuss the results for each of the datasets separately.

5.7.1 Columbus Large Image Format (CLIF)

A subset of the camera 0 Columbus Large Image Format (CLIF) described in Section 4.2.4 has been chosen. Non-objects sequences were created at random locations with a chip size equivalent to actual objects in each image sequence. The only constraint in the creation of these sequences was that the ground truth locations could not coincide. The CLIF dataset was used solely for binary object detection, with the only two classes being “object” and “non-object.”

Due to the intra-class variability of feature and invariant moment descriptors, edge histogram descriptors were the only descriptors evaluated. The results for object/non-object classification using only edge histogram bin counts of radii $r = 1,3,5,7,10,12,15,17,20$ and histogram statistics: mean, standard deviation, kurtosis, and skewness at each wavelet approximation level were obtained. All wavelet levels were combined to search for improvements. Figure 5.8 provides a plot of detection accuracies for each classifier at each wavelet level and at the additional ‘ALL’ level. The ‘ALL’ level produced the highest binary classification accuracy (88.52%) from the PART algorithm. For individual wavelet levels, level-1 produced the highest classification accuracy. This accuracy decreased as the wavelet level increased.

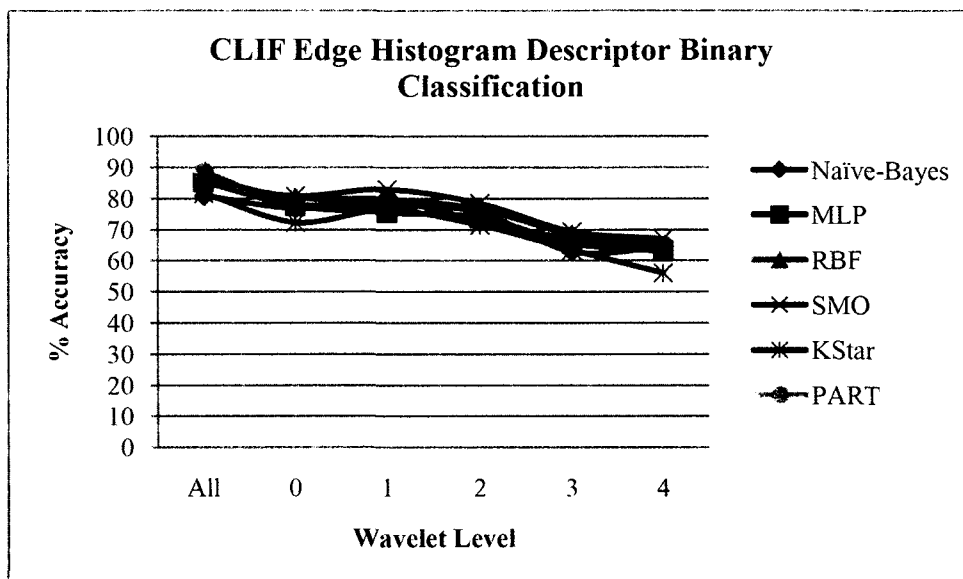


Figure 5.8 CLIF Object Detection Accuracy Using Edge Histograms

5.7.2 Army Research Lab CEGR Infrared Dataset

This dataset contains MWIR image sequences of ten intelligence objects travelling in a 100-meter circle from a range of 500 meters. Object chips were taken every ten frames as objects traveled horizontally in both directions. Eight of the ten objects were chosen for this study due to their relative similarity in size and structure. The object types were: Truck, SUV, BTR70, BRDM2, BMP2, T72, ZSU23-4, and 2S3. The Cincinnati Electronics Night Conqueror MWIR imager was combined with a Great River frame grabber to extract data. The Night Conqueror camera uses a 640 x 480 pixel Indium Antimonide (InSb) focal plane array (FPA) with 28-micron pitch. A system, which had a fixed FOV 300 mm lens resulting in a 3.4 x 2.6 FOV and a CO₂ notch cold filter, has been used. Multi-object type classification evaluation was performed using this dataset to study the discrimination accuracy of feature descriptors with different classification algorithms described in Section 5.6.1-5.6.5.

Here, the results obtained using the descriptors and classification methods described in Section 5.6 on the CEGR dataset where descriptors were calculated for object image chips (120 x 120 pixel size) are presented. The results for multi-class object classification using edge histogram bin counts of radii $r = 5, 10, 15, 20, 25, 30, 35, 40, 45, 50$ and histogram statistics: mean, standard deviation, kurtosis, and skewness at each wavelet approximation level are also shown. The plots in Figures 5.9 through 5.11 show the accuracy results for each descriptor type using wavelet levels $N = 0, 1, 2, 3, 4, 5$ as input. The descriptor set that contains all of the wavelet levels is denoted as the 'ALL' level. The results achieved after combining all of the descriptors for each wavelet level are presented in Figure 5.12. The best performance for the individual sets of the descriptors was the combined wavelet-level invariant moment descriptors, which gave a 94.14% classification accuracy using the KSTAR algorithm, and an 89.83% accuracy using MLP. The combined wavelet level for the edge histogram descriptors performed next highest with top accuracy results given by PART at 88.56%. The texture descriptors had the lowest accuracy of the feature sets, with the KSTAR classification method achieving 86.38% accuracy for all wavelet-level texture descriptors. When all descriptors were used together, the 'ALL' wavelet descriptors produced the highest accuracy, with SMO producing 98.55% accuracy and KSTAR and MLP producing 97% accuracy. The wavelet approximation level-1 had the highest individual accuracy, which then decreased as the wavelet level increased. The KSTAR and MLP algorithms provided the highest overall classification accuracy for CEGR.

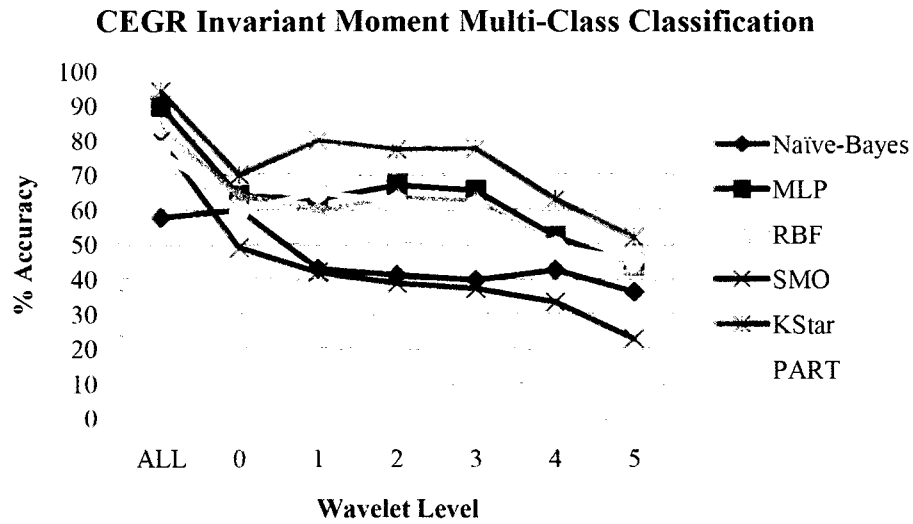


Figure 5.9 CEGR Classification Results Using Moment Invariants

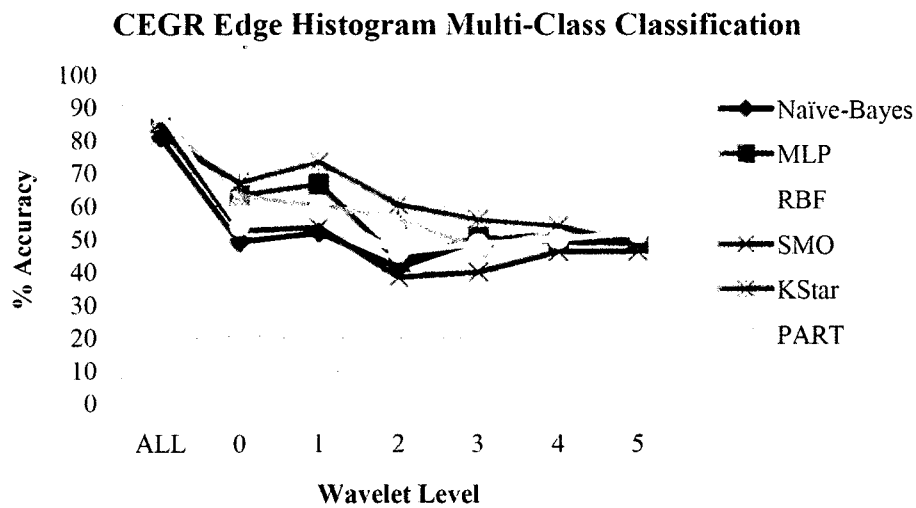


Figure 5.10 CEGR Classification Results Using Edge Histogram

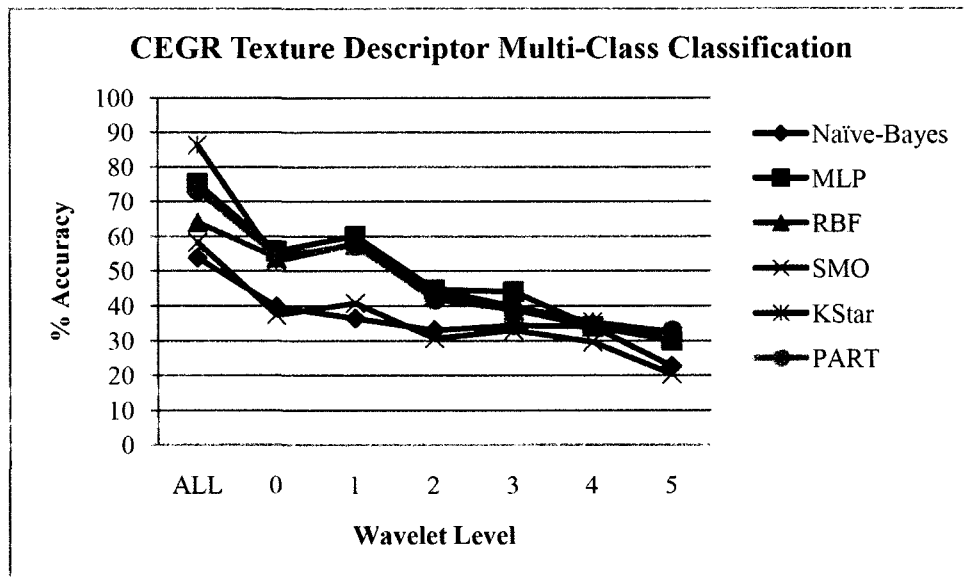


Figure 5.11 CEGR Classification Results Using Texture Statistics

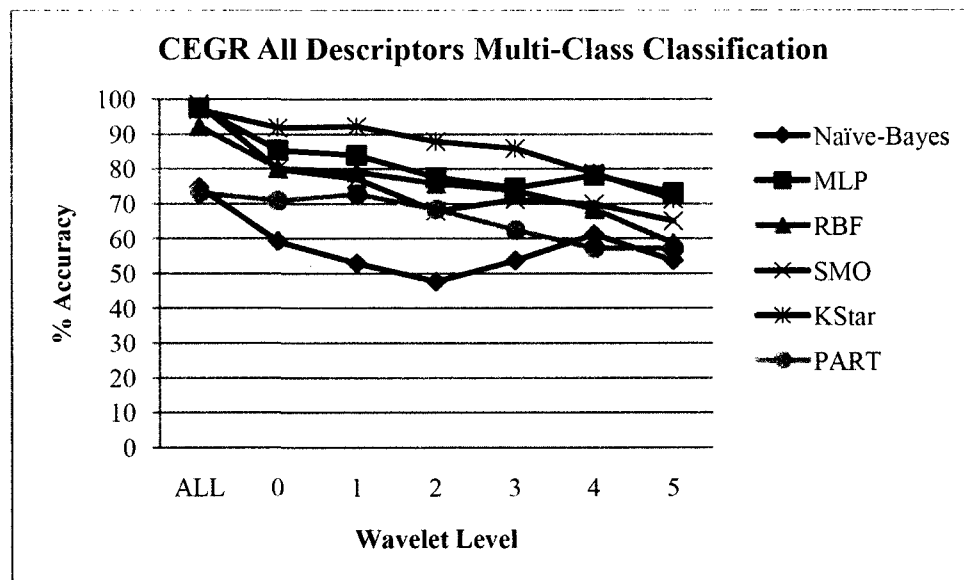


Figure 5.12 CEGR Classification Results Using Invariant Moments, Edge Histograms, and Texture Statistics

5.7.3 Army Research Lab IICO Visible Dataset

This dataset also contains each of the moving objects described in Section 5.7.2, with the same motion and range as the CEGR data. The visible light imagery was collected using a camera manufactured by Illunis that was referred to as IICO in the NVESD nomenclature. A Nikon zoom lens was adjusted to produce a 3.4-degree HFOV and locked in position. The output imagery was collected using a Coreco framegrabber. As with the CEGR dataset, multi-object type classification evaluation was performed using the IICO dataset to study the discrimination accuracy of feature descriptors with different classification algorithms as described in Section 5.7.2.

In this section, the results obtained from using the descriptors and classification methods described in Section 5.7.2 on the IICO dataset are presented. In this experiment, descriptors were calculated for object image chips (120 x 120 pixel size). The plots in Figures 5.13 through 5.15 show the accuracy results for each descriptor type using wavelet levels $N = 0,1,2,3,4,5$ as input, and the descriptor set that contains all of the wavelet levels is denoted as the 'ALL' level. Results (in Figure 5.16) received when all of the descriptors for each wavelet level are described. The best performance for the individual sets of the descriptors was the combined wavelet-level invariant moment descriptors, which achieved a 92.86% classification accuracy using the KSTAR algorithm, and an 86.07% accuracy using SMO. The combined wavelet level for the edge histogram descriptors (using the same bins and statistics as CEGR) performed next highest with top accuracy results given by MLP at 89.64% and KSTAR with 87.86%. The texture descriptors again had the lowest accuracy of the feature sets, with the MLP classification method giving 70.71% accuracy for all wavelet-level texture descriptors.

When all descriptors were used, the “ALL” wavelet levels produced the best results with 98.57% accuracy using the KSTAR algorithm, with SMO producing a 96.43% accuracy and MLP producing a 95.36% accuracy. Level-0 (original image) produces the best results for individual wavelet levels for all descriptors with decreasing accuracy correlating with increasing wavelet levels. The level-1 wavelet approximation coefficients produced the highest accuracy results for moment invariant descriptors, and wavelet level-2 coefficients performed the best for edge histogram descriptors. All of the wavelet levels performed roughly the same for texture descriptors. As with CEGR, the KSTAR and MLP algorithms provided highest classification accuracy overall for the IICO dataset.

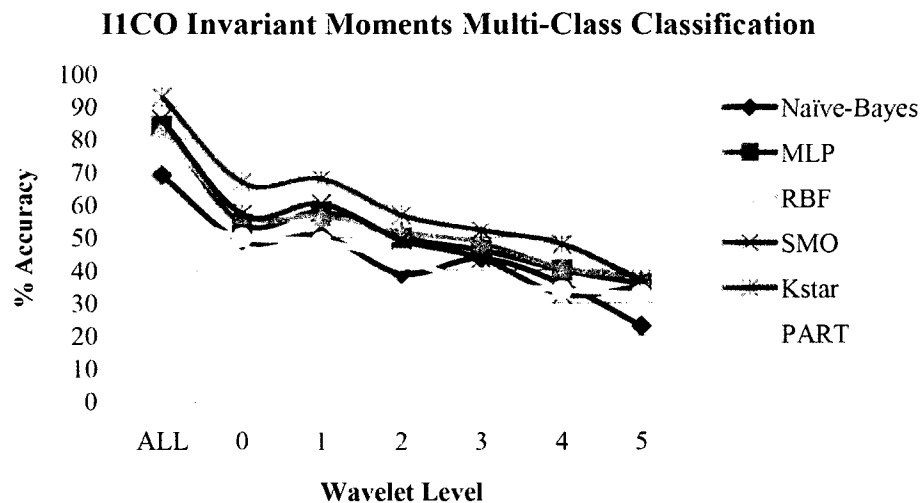


Figure 5.13 IICO Classification Results Using Moment Invariants

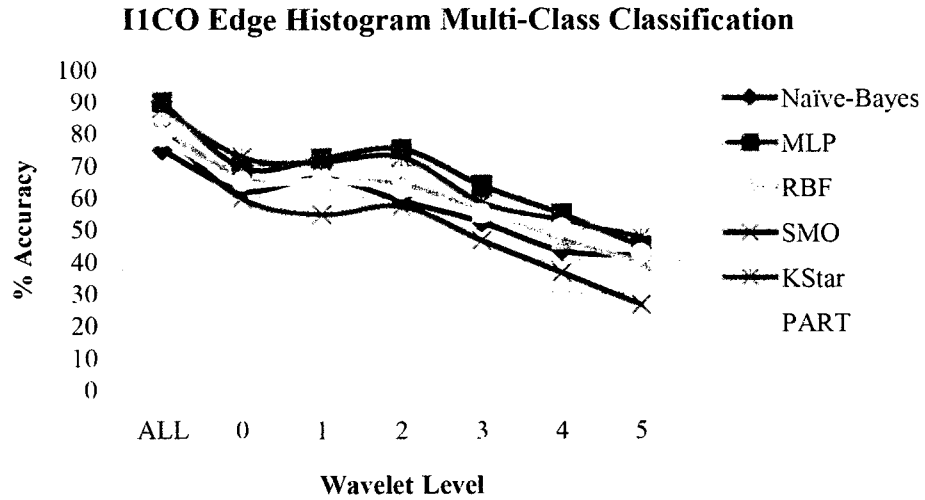


Figure 5.14 IICO Classification Results Using Edge Histogram Statistics

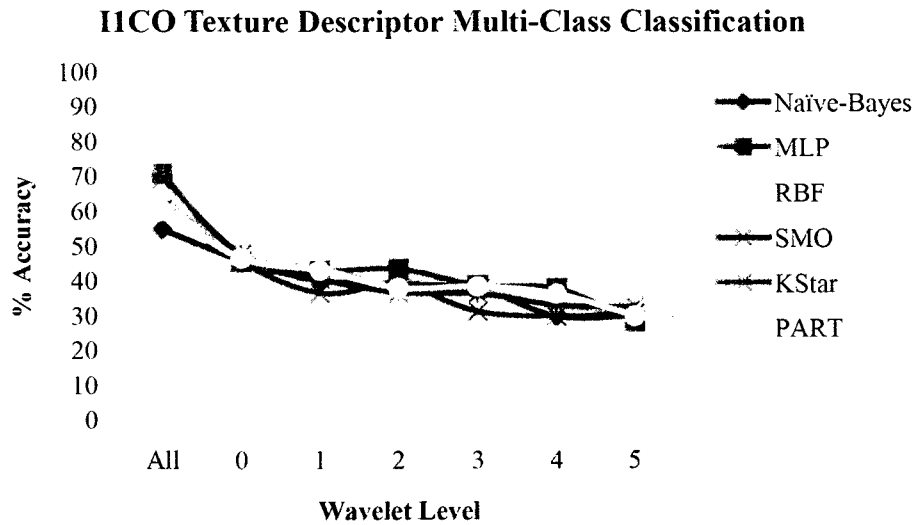


Figure 5.15 IICO Classification Results Using Texture Statistics

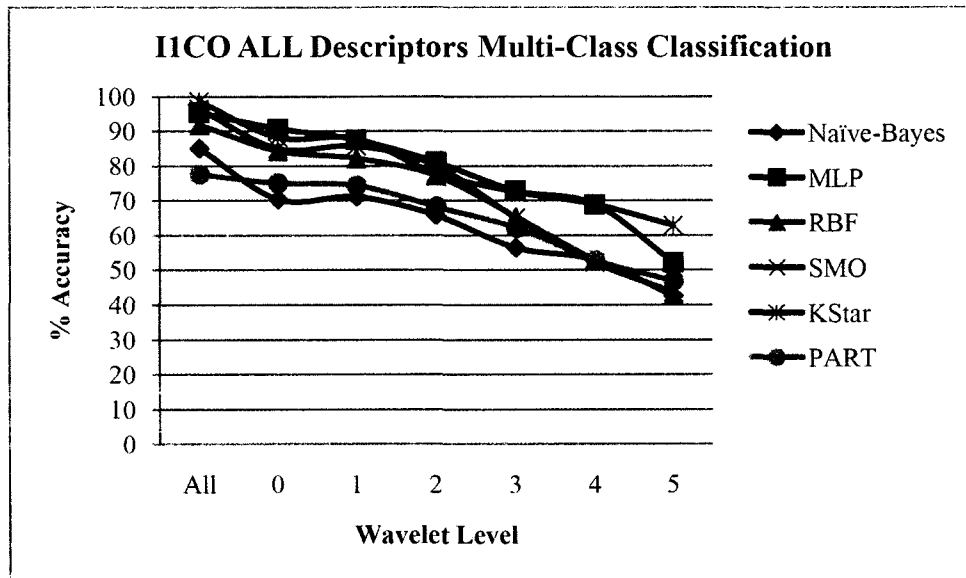


Figure 5.16 IICO Classification Results Using Invariant Moments, Edge Histograms, and Texture Statistics at Wavelet Levels $N = 0, 1, 2, 3, 4, 5, \text{ALL}$.

5.8 Feature Extraction and Classification Conclusion

The feature descriptors that provide discrimination for classification using varying levels of wavelet approximations of object image chips have been extensively tested. A multi-scale wavelet representation provides the best input for calculating object descriptors, with best results occurring with wavelet levels 1 and 2 for all three datasets. Invariant moments and edge histogram statistics out-performed the texture statistics for both the IICO and CEGR datasets in classification accuracy for all the classifiers used. KSTAR and MLP algorithms consistently performed better than the remaining algorithms, with Naïve-Bayes consistently producing the lowest accuracy. Optimum performance for multi-class classification for objects with high resolution is to combine invariant moment and edge histogram descriptors using wavelet 0-2 levels. For object detection (binary classification) for low-resolution images with large intra-object

intensity variability like the CLIF dataset, edge histogram statistics generated from multi-scale (0-2) wavelet approximation coefficients provides an accurate solution for object detection.

CHAPTER 6

TRACKING FOR SE

6.1 Introduction

As discussed in Chapter 2, several classes of object tracking algorithms are commonly found in the literature. Tracking methods based on previous information used in detection and classification (Figure 6.1) have been designed.

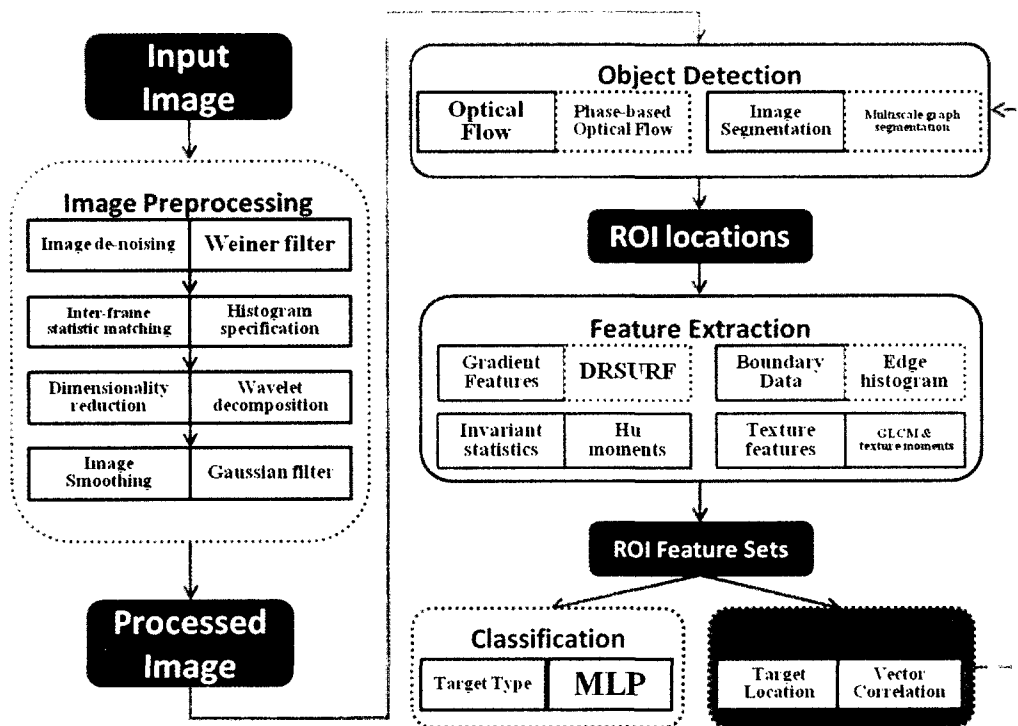


Figure 6.1 Object Tracking in SE Framework

The feature tracking methods can use both optical flow and DRSURF feature descriptors to create a fused set of descriptors, which are then compared to values from the detected locations in the new image sequence. A description of how to find optical flow feature and DRSURF feature tracking values is presented in Section 6.2.

6.2 Optical Flow Object Tracking

After optical flow is calculated between two consecutive images, optical flow at known object locations are measured by calculating an average of the absolute value of the top n optical flow responses in horizontal and vertical directions, $|\mu_{n_{max}}| = \frac{|\mu_{n_x}| + |\mu_{n_y}|}{2}$ in a window of size w_t based on the object bounding box. If $|\mu_{n_{max}}| > thresh_v$, a velocity threshold set to the smallest optical flow value observed when an object centroid has moved to a pixel location larger than 3, the windowed object search algorithm is initialized. An object search field is chosen to include $2v_{max} \times 2v_{max}$ windowed area centered at the object location. Each location $|\mu_{field_{n_{max}}}| > thresh_v$ is chosen as a candidate for a new object location. After studying the optical flow in regions of tested object tracks, a high correlation has been found between previous and current object locations through the optical flow using the following negative vector correlation function:

$$corr_{iOF} = -\frac{[OF_{iold}] \cdot [OF_{inew}]}{\sqrt{[|OF_{iold}|] \cdot [|OF_{iold}|] * [|OF_{inew}|] \cdot [|OF_{inew}|]}}, \quad \{i = x, y\}, \quad (6.1)$$

where OF_{iold} and OF_{inew} are the vectorized forms of the 2-D optical flow values of size $w_t \times w_t$ for both the horizontal and vertical directions. The top value means are also included in the correlation vector, written as:

$$corr_{\mu_{nmax}} = \frac{[\mu_{old\ nmax}] \cdot [\mu_{new\ nmax}]}{\sqrt{([\mu_{old\ nmax}] \cdot [\mu_{old\ nmax}]) \cdot ([\mu_{new\ nmax}] \cdot [\mu_{new\ nmax}])}} \quad (6.2)$$

These two correlation values are then used to find the location with highest correlation, using a weighted combination of:

$$corr_{total} = [corr_{xOF} \quad corr_{yOF} \quad corr_{\mu_{nmax}}] \cdot [.25 \quad .25 \quad .5]^T. \quad (6.3)$$

For detection, it is assumed that there is no prior knowledge of a specific object of interest, so we measure only the $|\mu_{nmax}|$ at each central pixel location in a windowed region, w_t , which is set to the average object bounding box size.

An object is considered successfully tracked to a sequential image frame if the object ground truth centroid and tracker algorithm selection point are within half the distance of the object-bounding box, w_t . Figure 6.2 shows results obtained from an optical flow tracking algorithm for moving objects at each wavelet level.

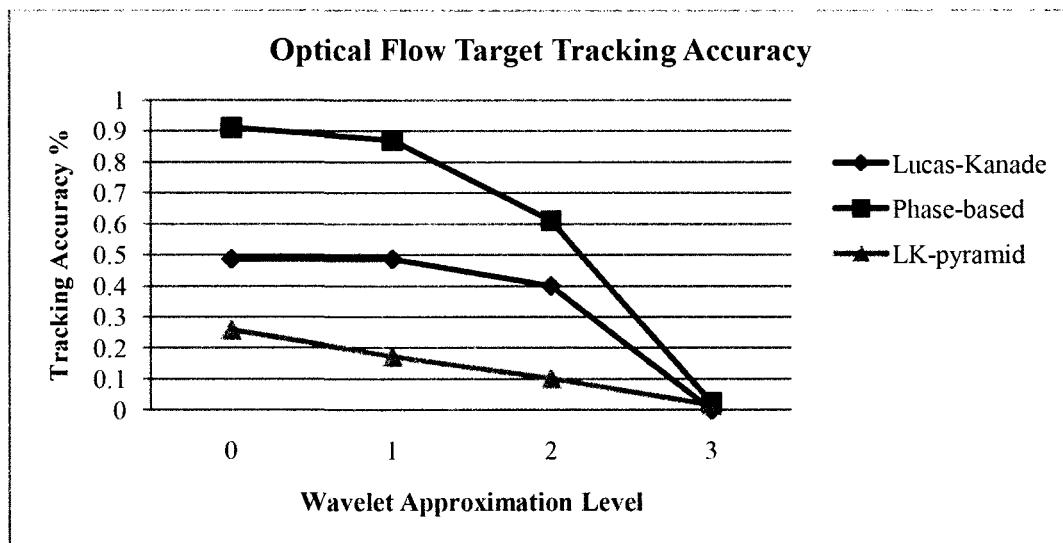


Figure 6.2 Object Tracking Accuracy Results for Each Optical Flow at Wavelet Levels $N=0,1,2,3$

Optical flow can increase the accuracy and specificity of object tracking methods, and a combination of optical flow and feature tracking methods can produce the best results for robust aerial object tracking.

6.3 DRSURF Feature Tracking

The DRSURF feature calculation process detailed in Section 5.2 is used as an illumination invariant feature set for tracking using vector correlation similar to the optical flow tracking. The implementation for tracking and provide an evaluation is described.

6.3.1 DRSURF Feature Tracking Implementation

An object chip is provided to the tracker for the creation of the DRSURF feature descriptor vector, and its location in the image is noted. Next, the optical flow is taken between the image with known object and location and a new image frame with unknown object position. The search space is reduced to a window surrounding the last known object location of size $2v_{max} \times 2v_{max}$, where v_{max} is the previously determined maximum object velocity. This area is further reduced to locations where the optical flow is larger than a previously determined threshold for minimum motion detection. The DRSURF features are calculated for the remaining locations at orientations $0^\circ, -15^\circ, 15^\circ, -30^\circ, 30^\circ$ with respect to the previously found object. A vector correlation similarity measure is then employed to quantify the similarity of the candidate descriptor vectors at each orientation, $\mathbf{d}_{feild_{orient}}$, with the object descriptor, \mathbf{d}_{targ} , using the following equation:

$$corr_{targ-feild} = \frac{[\mathbf{d}_{targ}] \cdot [\mathbf{d}_{feild_{orient}}]}{\sqrt{([\mathbf{d}_{targ}] \cdot [\mathbf{d}_{targ}]) * ([\mathbf{d}_{feild_{orient}}] \cdot [\mathbf{d}_{feild_{orient}}])}}. \quad (6.9)$$

The maximum correlation location and orientation are chosen as the new object location and new orientation for the next frame. The optical flow of the next frame is then calculated, and the process is repeated. If optical flow is not found at the object location, the tracker retains the same object location for the frame.

6.3.2 Feature Tracking Evaluation and Results

For testing DRSURF feature tracking, a subset of the camera 0 Columbus Large Image Format (CLIF) sequence as described in Section 4.2.2 is used. For object tracking, the methods outlined in Section 5.2.4 are used. An object is considered successfully tracked to a sequential image frame if the object ground truth centroid and tracker algorithm selection point are within half the distance of the object bounding box, w_t . Figure 6.3 gives the tracking accuracy of varying DRSURF feature lengths, which is accompanied by their execution times in Figure 6.4. Figure 6.5 shows the results of an optical-flow / feature-tracking algorithm using DRSURF feature descriptors, SURF descriptors, object histogram, pixel intensities, and invariant moments. Tracking accuracy is calculated at varying levels of optical-flow sensitivity, with a lower threshold increasing possible candidate locations the feature-tracking step.

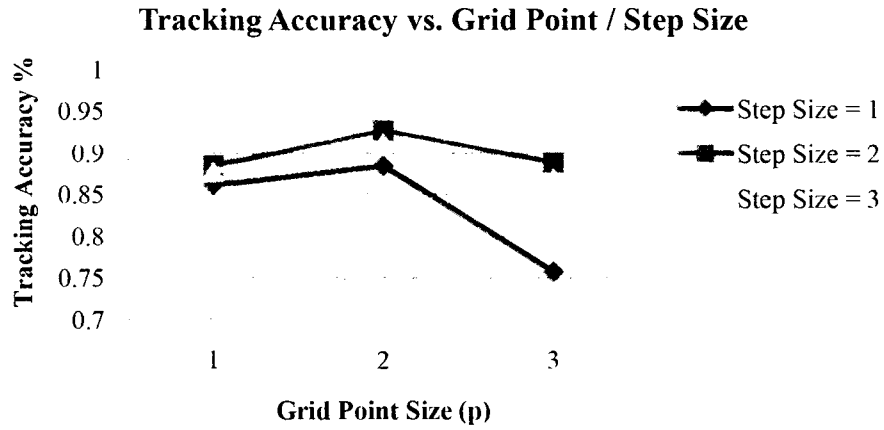


Figure 6.3 DRSURF Feature Tracking Accuracy for Varying Feature Lengths

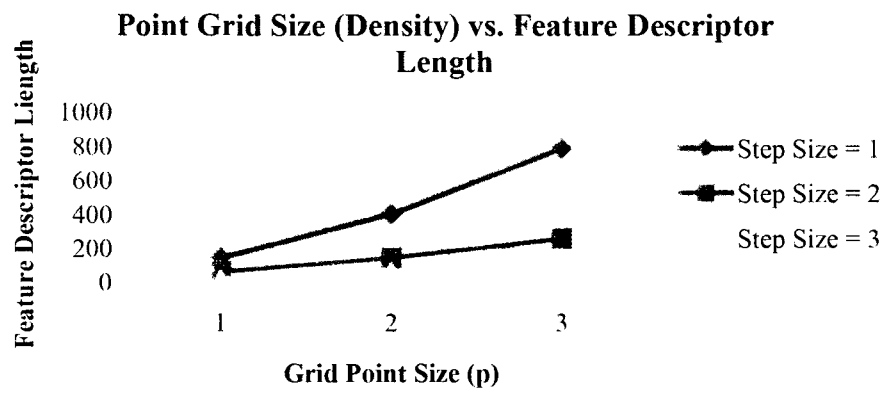


Figure 6.4 DRSURF Feature Tracking Execution Time for Varying Feature Lengths

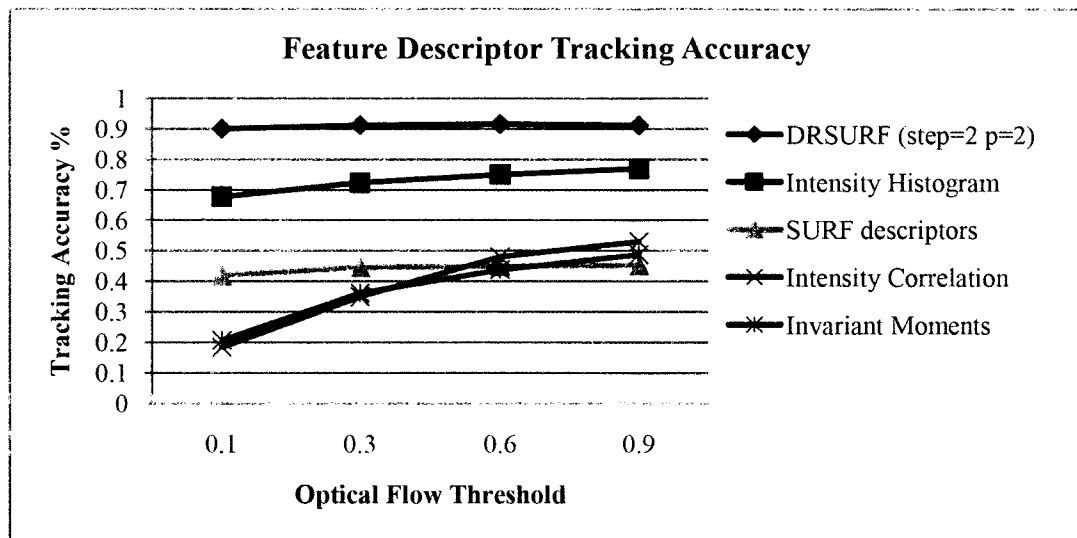


Figure 6.5 Tracking Accuracy Results for DRSURF and Other Methods at Different Optical Flow Thresholds

6.3.3 DRSURF Feature Tracking Conclusion

The DRSURF feature tracker outperformed (91% average accuracy) the other descriptor values at each of the optical-flow thresholds given in Figure 6.5. Unlike methods used to test other feature sets, the DRSURF tracking method remains relatively unchanged at the different optical flow thresholds, providing evidence that the method works over large fields of potential object locations (high specificity), without loss of accuracy. The DRSURF method is a viable solution for low-dimensional object tracking in aerial image sequences due to its dense and robust feature vector. Although the multi-orientation calculations results in increased computation, the increased accuracy provides the trade off in time, especially because the method can now be parallelized for enhanced performance.

6.4 Object Tracking Conclusion

Two feature tracking methods which depend on a correlation score for updating the SE with new locations of objects have been presented and evaluated. This method has shown to be successful in the data sets, with accurate tracking for both translation and rotational motion. The data sets lack strong affine variations that some researchers in SE may be interested, which causing large variations in object statistics that correlation tracking may have problems handling. To combat such challenges, cascading tracking methods that employ multiple trackers simultaneously to create a higher-level decision-metric that weights tracker certainty based on prior tracker performance have been studied. The current methods would incorporate well in such a scheme, especially if it is coupled with a dynamic state modeling function such as a particle filter, due to the formers ability to accurate follow non-occluded motion and latter's ability to handle frames when the object is hidden.

CHAPTER 7

GPU IMPLEMENTATION OF THE SE FRAMEWORK

7.1 GPU Introduction

A relatively cheap and powerful parallel processing platform is using a workstation's graphical processing units (GPU). Now that several computer vision algorithm implementations using GPUs have been developed over the past five years [84][85][86], implementing SE methods for GPU processing requires limited GPU-specific software development experience.

Although many steps in SE are performed sequentially (output in previous step is input in the next), particular tasks can benefit from parallel processing due to their independent calculations. Due to the extra CPU to GPU communication overhead incurred when using GPUs, each stage of the SE workflow should be evaluated to determine the execution time gains.

In this chapter, the GPU implementations of the SE framework using varying data dimensions are presented, evaluated, and discussed. The reader is provided with recommendations on when to use GPU-based implementations for calculating steps for preprocessing, segmentation, motion estimation, feature extraction, and tracking functions for SE.

7.2 GPU-Based SE Framework Implementations

In this section, the SE framework is divided into the sequential steps that take multiple sensor data frames as input and return object location and track data. The workflow tackles each task using the previous stage outputs as an input, thus disallowing GPU implementation upon the entire algorithm. The computation required to accomplish each of the SE framework's tasks the GPU-enabled implementation for potential execution time gains are described. The execution time and speed up will be provided for each stage described in the following sections.

The performance of the GPU-based methods on representative inputs of varying sizes with a CPU-based implementation is evaluated. The GPU-based implementation of a single calculation is used, and parallel iterations are performed when applicable to the particular task. The inputs used on each task are comparable to typical dimensions based on the original input data (in the case 2-D images with 0-50 objects present). Each data input and iteration benchmark size are performed 200 times, using the average time per run as the benchmark in Figures 7.1-7.30.

The CPU used in the following experiments is a single-core of the 2.67 GHz Intel i7 920 processor with 12 GB of RAM. The GPU used in the testing is the Tesla C1060 GPU with 4095 MB VRAM. Benchmarking code was written in MATLAB using Jacket wrapping functions and benchmarking functions found in reference [87].

7.2.1 GPU for Image Preprocessing

As described in Section 2.2, most SE methods initially perform denoising and smoothing operations to the input data in aims to reduce irregularities in local neighborhoods. A popular method, which convolves a 2-D Gaussian kernel with input

images (Section 3.5), is implemented. Due to the potentially large dimensionality of the input image when compared with the typical Gaussian kernels used ($\approx 5 \times 5$ pixels), the independent operations of a 2-D convolution lends itself to potential execution time gains through GPU parallel processing.

In Figures 7.1 and 7.2, the CPU-GPU execution times and speed-up gains are shown, respectively, varying the input size to image matrices of 10-2900 pixels per dimension. As can be seen from both figures, GPU implementation outperforms CPU past 300 pixels per dimension, reaching 4x speed gains 1000 pixels.

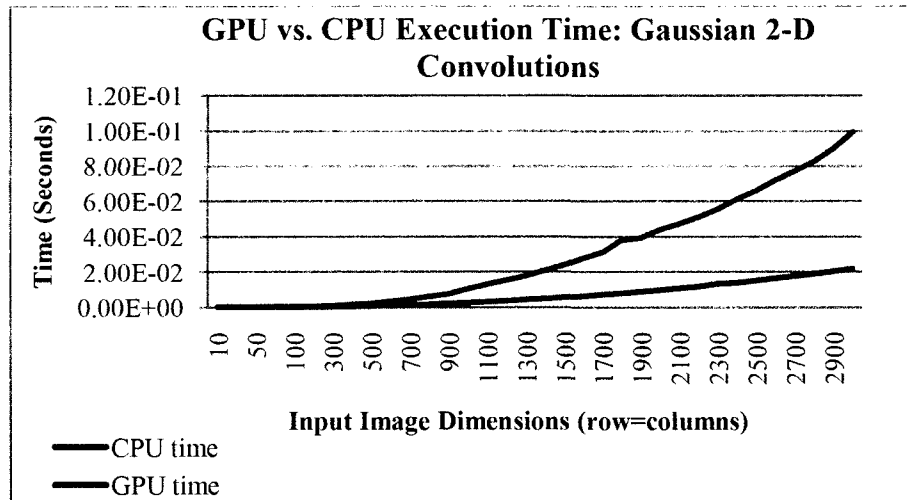


Figure 7.1 GPU-CPU Execution Times for 2-D Gaussian Convolutions

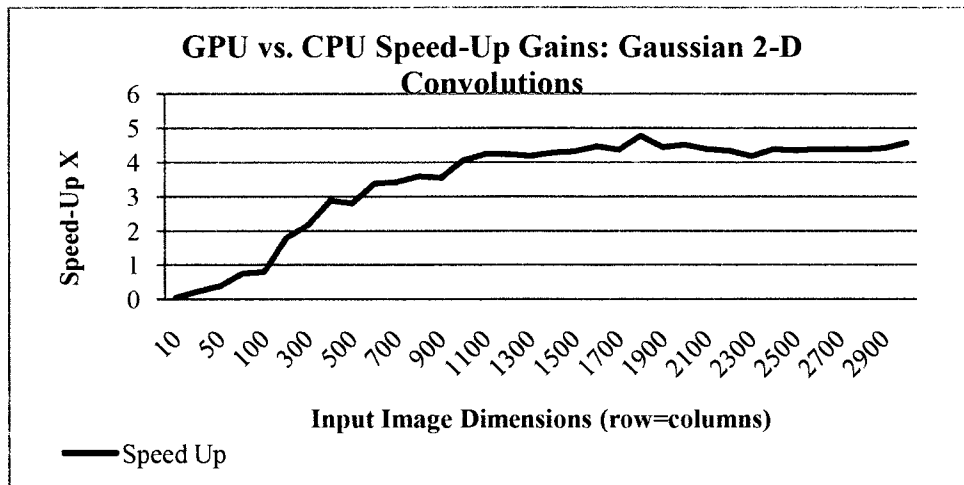


Figure 7.2 Speed-Up Gains Using GPU for Gaussian 2-D Convolutions

7.2.2 GPU for Object Detection

As described throughout Chapter 4, motion detection is a difficult and computationally cumbersome problem, especially dealing with increasingly large sensor data.

Performance gains in execution time using GPUs for both optical flow for motion estimation and graph segmentation for object detection are used.

As described in Section 4.2.3, phase-based optical calculates a bank of Gabor filters to estimate velocity. Equation (4.15) is used to generate these filter responses, which can be performed independently. A GPU can perform the set of 1-D convolutions for each Gabor filter simultaneously for the entire filter bank, potentially gaining essential speed-up time. Figures 7.3 and 7.4 show these results of Gabor filter bank calculations for varying input image dimensions from 10-500.

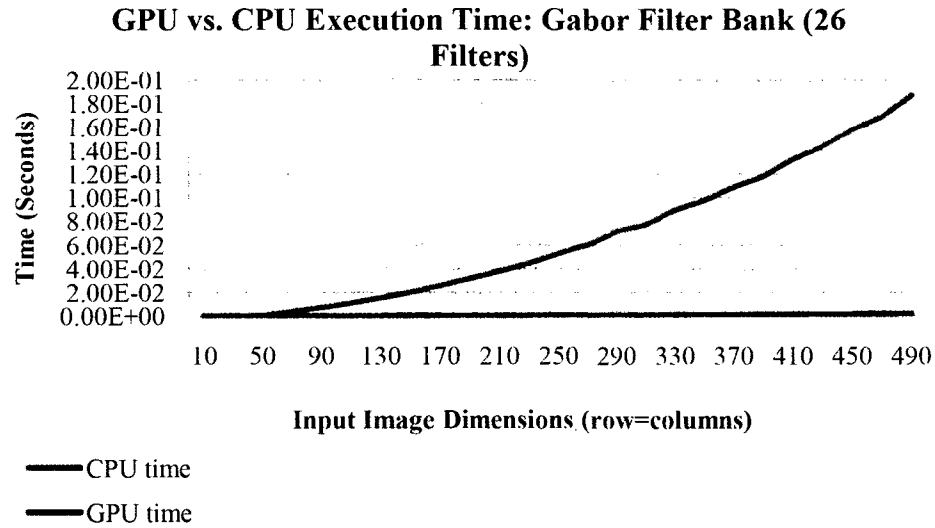


Figure 7.3 GPU-CPU Execution Times for Gabor Filter Bank

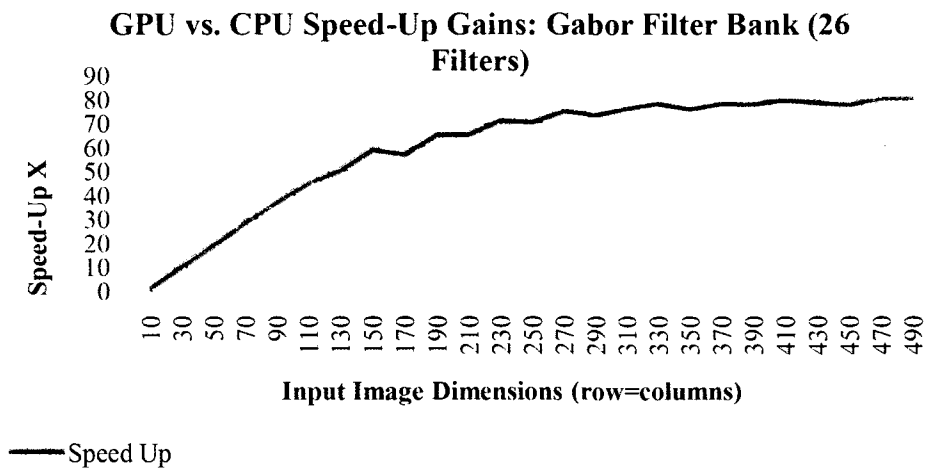


Figure 7.4 GPU-CPU Speed Up for Gabor Filter Bank

GPU implementation for Gabor filter bank calculations outperforms throughout the entire input size, which is expected due to the independent computations reducing execution times.

As discussed in Section 4.3, multi-scale graph segmentation provides a hierarchical set of segmentations that contain accurate object segments. This process requires both large computations due to the vectorization of the 2-D input, as well as an iterative node agglomeration. Any speed gains that can improve execution times allow for larger input values to be used in the method. The graph initialization stage requires significant completion time when performed iteratively, but as the calculations are performed independently in local neighborhoods (Equation 4.16 and 4.17), this step is implemented in the GPU. Figures 7.5 and 7.6 give the execution time and speed-up gains for varying input matrix sizes. An exponential speed-up gain for the GPU implementation, which is necessary for computing even medium-sized input matrices (due to the vectorization) is shown. The GPU speed-up surpasses 10,000 at 600 dimension size, which is a reasonably expected input size in some sensor data.

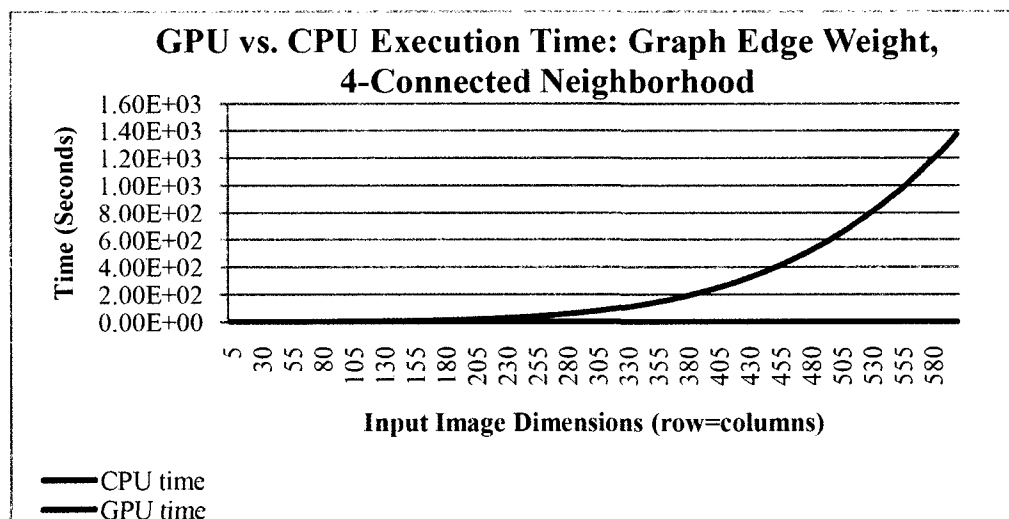


Figure 7.5 GPU-CPU Execution Times for Graph Edge Weighting

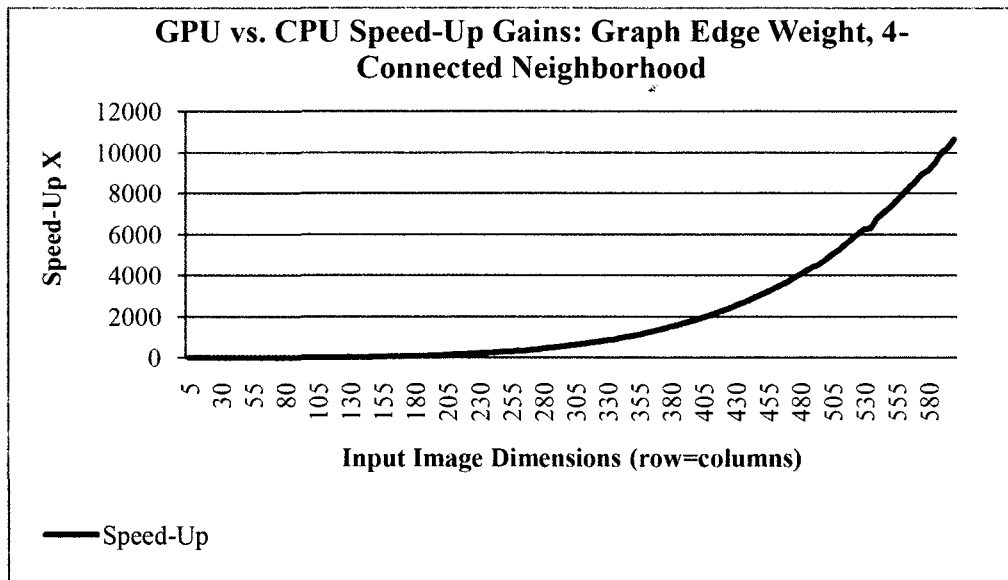


Figure 7.6 GPU-CPU Speed-Up for Graph Edge Weighting

As discussed in Section 4.3.2, nodes in the hierarchical graph clustering scheme are selected based in a normalized cuts-like fashion, where the generalized eigen problem (Equation 4.18) is solved. This gamma function is required of each node in the graph, thus independent calculations are performed. This recursive system continually performs this function on smaller sets of graph nodes, which will make the GPU implementation progressively less efficient until a node size is reached where CPU can perform the method more efficiently. A series of GPU-CPU execution time and speed-up gains (Figures 7.7-7.12) is used for varying node sizes and iterations, to clearly define breakeven points in the implementation speed-up gains. For varying input sizes, CPU operations outperform the GPU, although times are fairly comparable. CPU computations also outperform the GPU for smaller iteration sizes, but a GPU gain is observed for iterations over 7500 nodes.

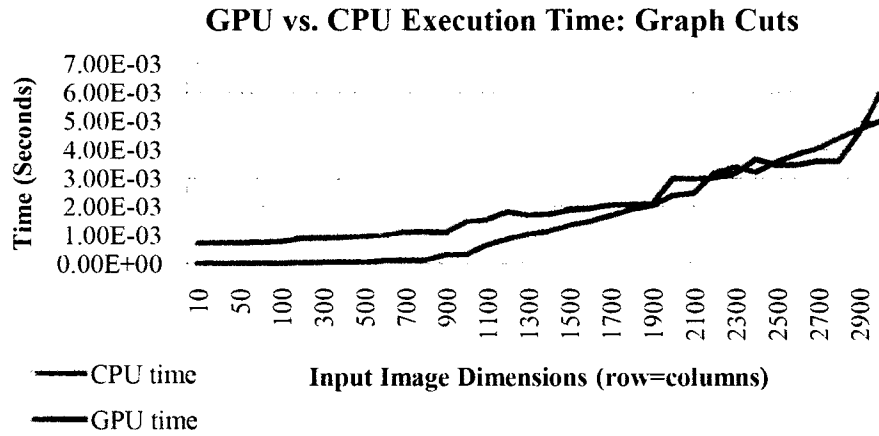


Figure 7.7 GPU-CPU Execution Times for Graph Cuts (no loops)

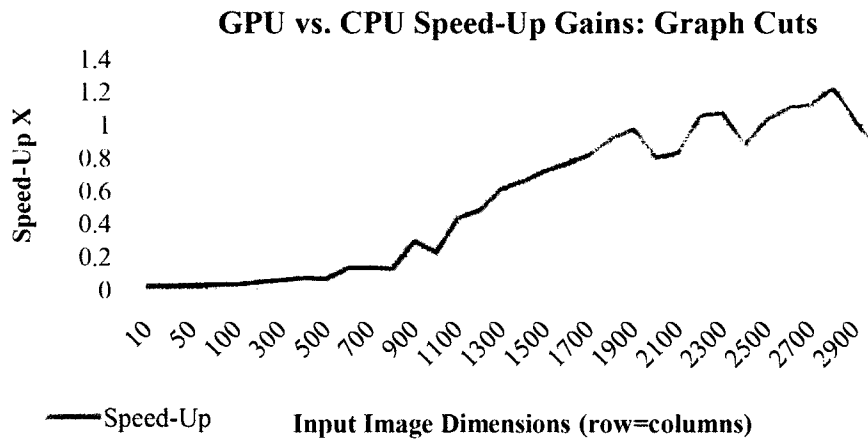


Figure 7.8 GPU-CPU Speed-Up for Graph Cuts (no loops)

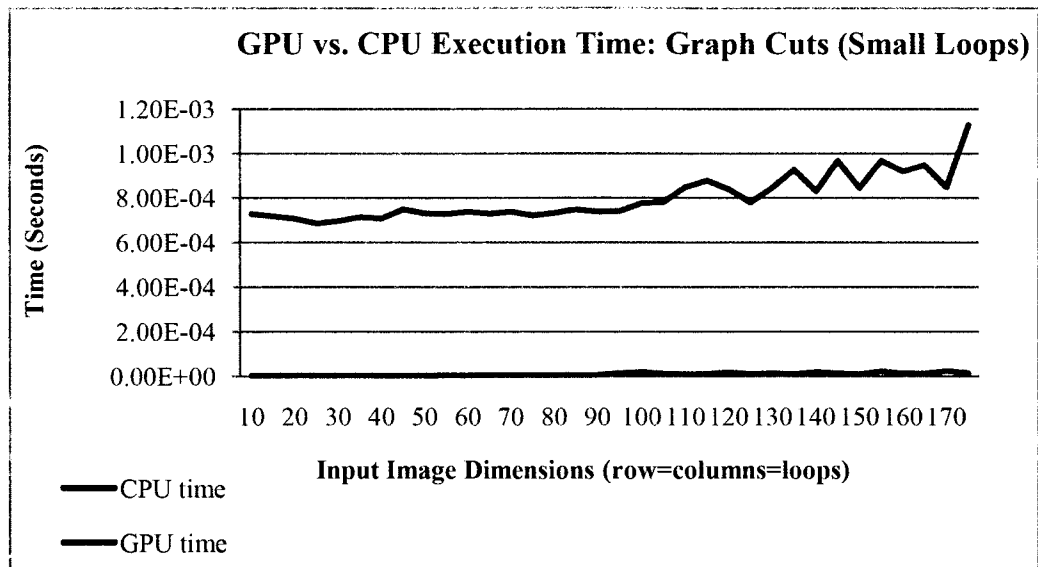


Figure 7.9 GPU-CPU Execution Times for Graph Cuts (small loops)

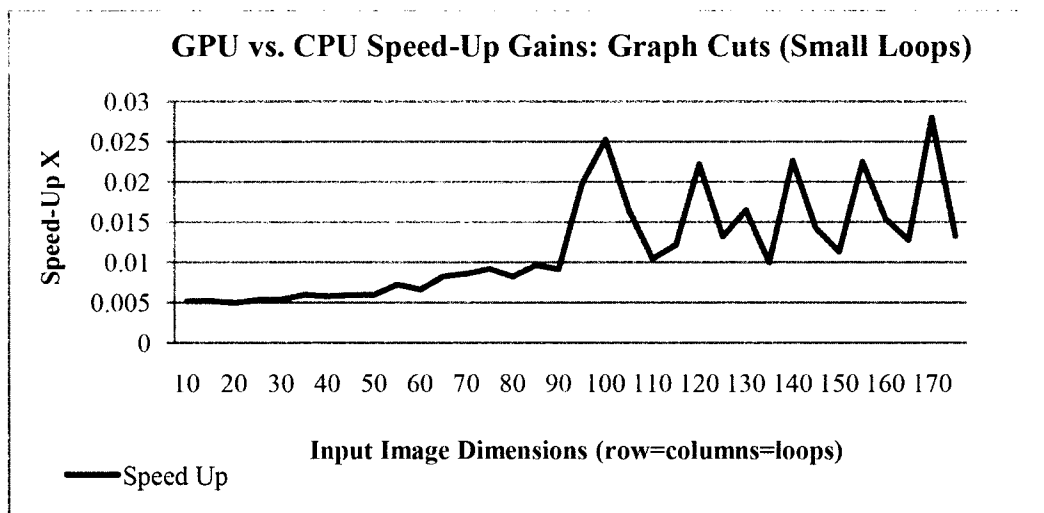


Figure 7.10 GPU-CPU Speed Up for Graph Cuts (small loops)

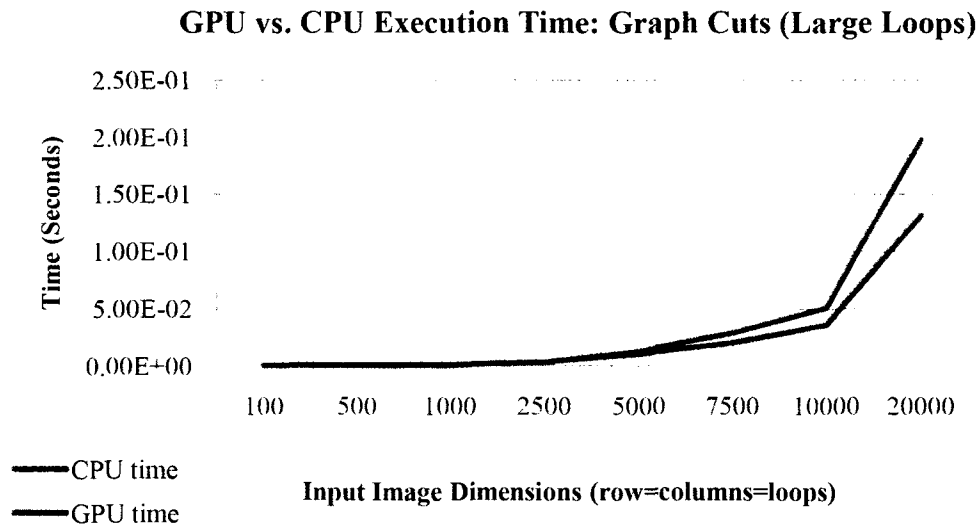


Figure 7.11 GPU-CPU Execution Times for Graph Cuts (large loops)

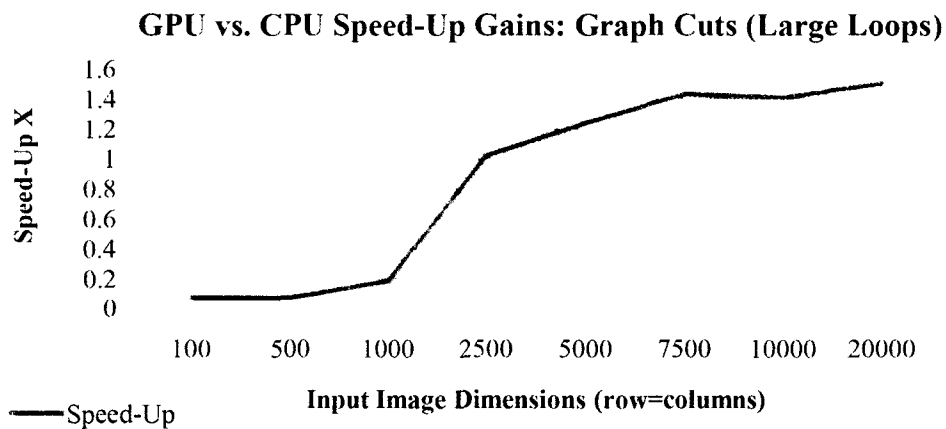


Figure 7.12 GPU-CPU Speed Up for Graph Cuts (large loops)

7.2.3 GPU for Feature Extraction

As described in Chapter 5, although the potential number of candidates is significantly lower than initial SE stages (preprocessing and motion estimation), a dense set of features for each object candidate at every location can now be calculated simultaneously, validating the economy of using a GPU implementation. GPU based

computations are implemented for both histograms and invariant moments in hopes to improve execution time.

In Section 5.4, the methodology of edge histogram features for object classification is discussed. This step requires several steps for computation, with a final step using a calculation similar to Equation (5.7). Each object candidate location requires an independent set of calculations for edge histograms, and GPU speed gains in similar fashion to graph cuts to see gains for specific input sizes (Figures 7.13.-7.19) are presented. When no iterations are performed, a constant, larger execution time for GPUs can be observed. For iterations, the GPU implementation outperforming the CPU for iteration sizes larger than 20, which is normally expected for real applications, are observed.

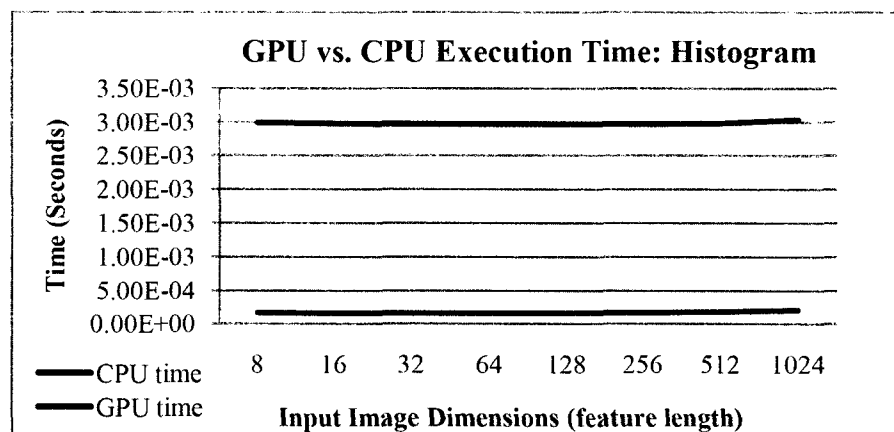


Figure 7.13 GPU-CPU Execution Time for Histogram (no loops)

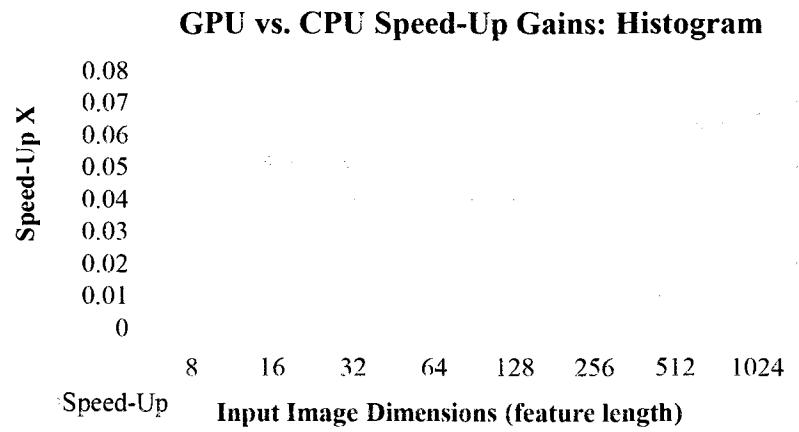


Figure 7.14 GPU-CPU Speed Up for Histogram (no loops)

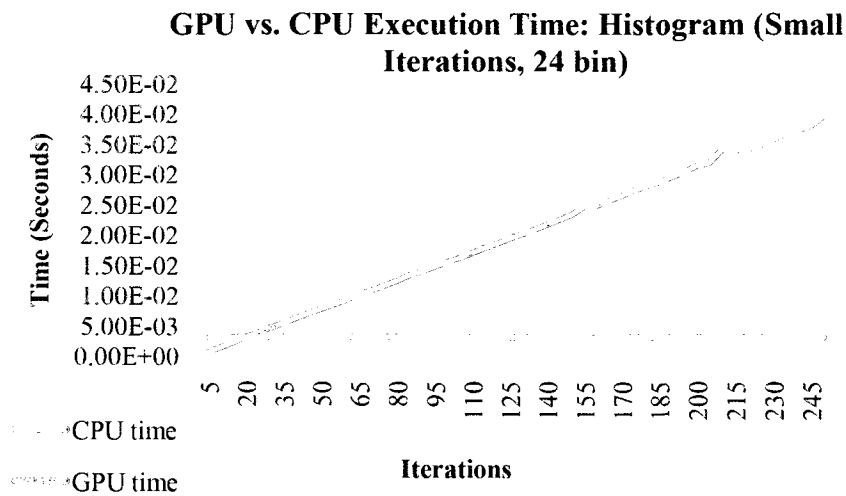


Figure 7.15 GPU-CPU Execution Time for Histogram (small loops)

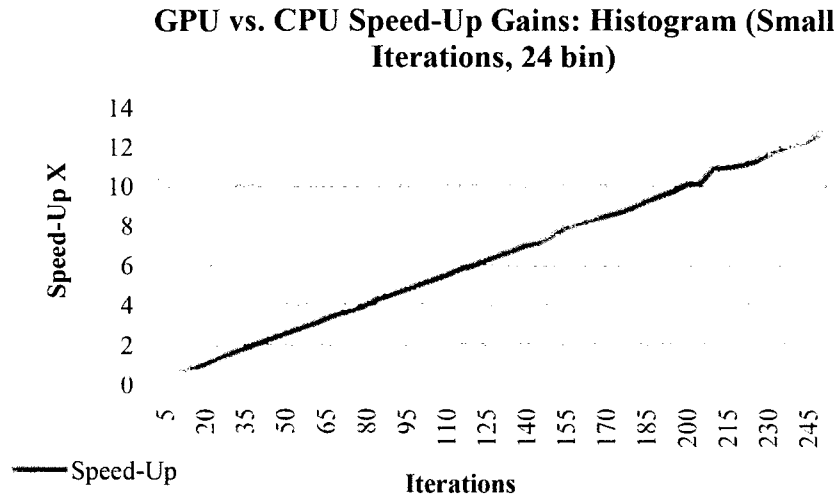


Figure 7.16 GPU-CPU Speed Up for Histogram (small loops)

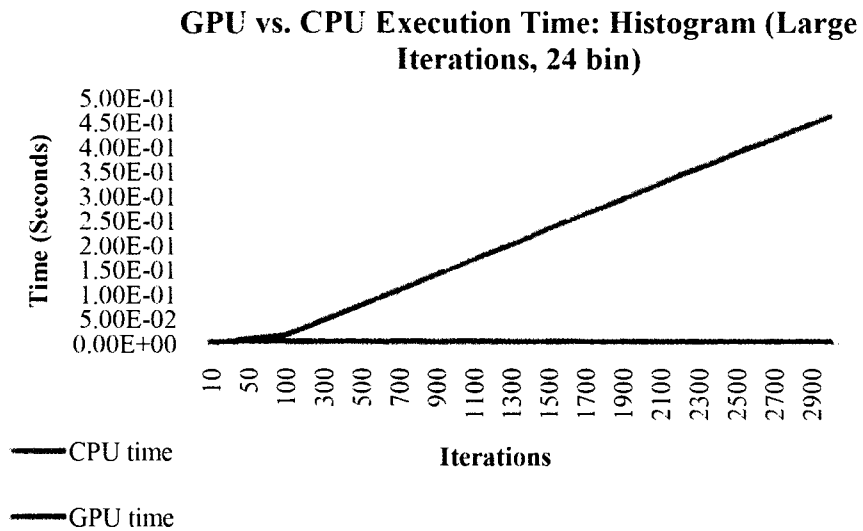


Figure 7.17 GPU-CPU Execution Time for Histogram (large loops)

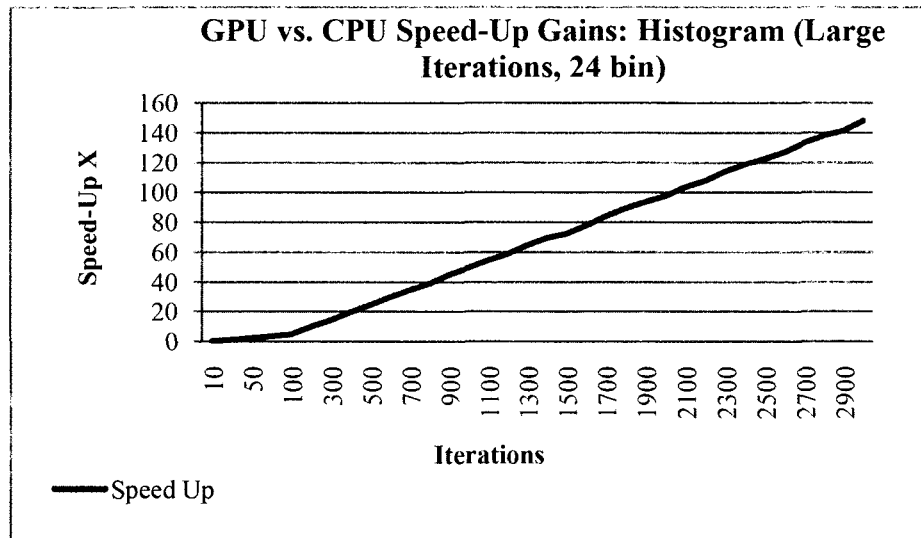


Figure 7.18 GPU-CPU Speed Up for Histogram (large loops)

The methodology and presented the implementation of invariant moments in Section 5.3, using this set of features in object classification, have been discussed. Similar to feature histograms, this feature set is also calculated in a local, independent manner for each object candidate, thus lending itself to speed-up gains via parallelization. In Figures 7.19-7.24, these results are presented. Unlike histograms, CPU-GPU execution times remain similar throughout a single calculation with varying input sizes. GPU speed-up gains are evident for the entire span of iterations, following a linear speed-up with increasing loop size.

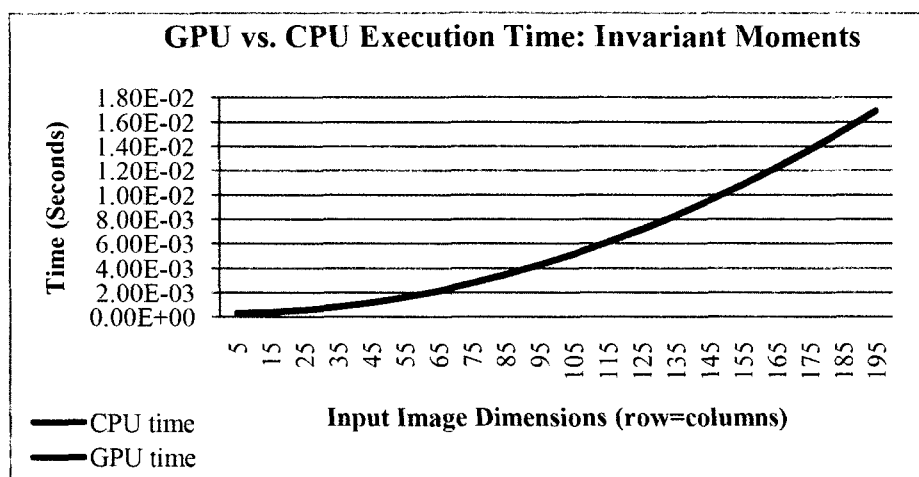


Figure 7.19 GPU-CPU Execution Time for Invariant Moments (no loops)

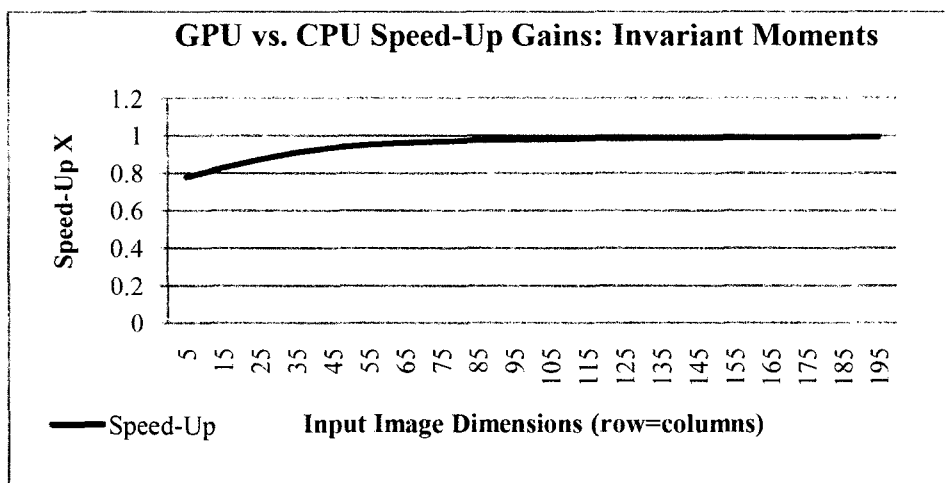


Figure 7.20 GPU-CPU Speed-Up for Invariant Moments (no loops)

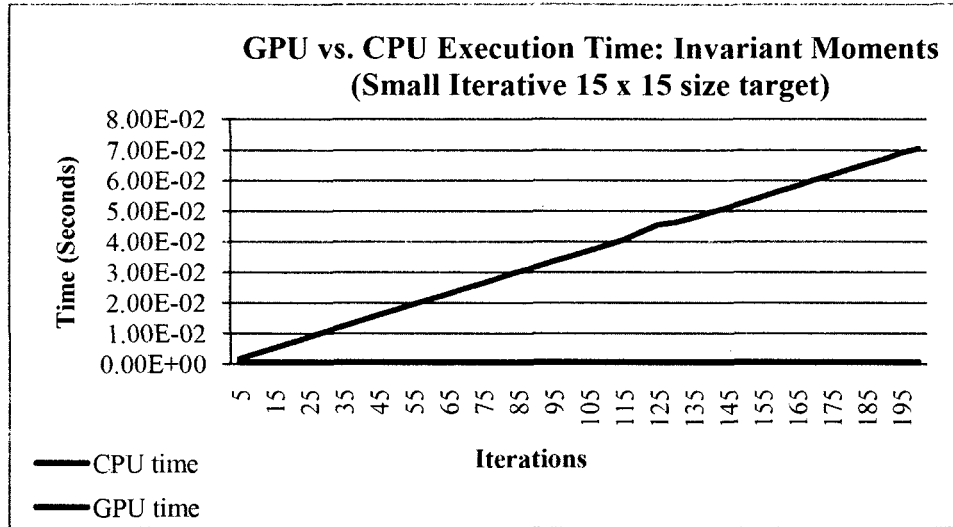


Figure 7.21 GPU-CPU Execution Time for Invariant Moments (small loops)

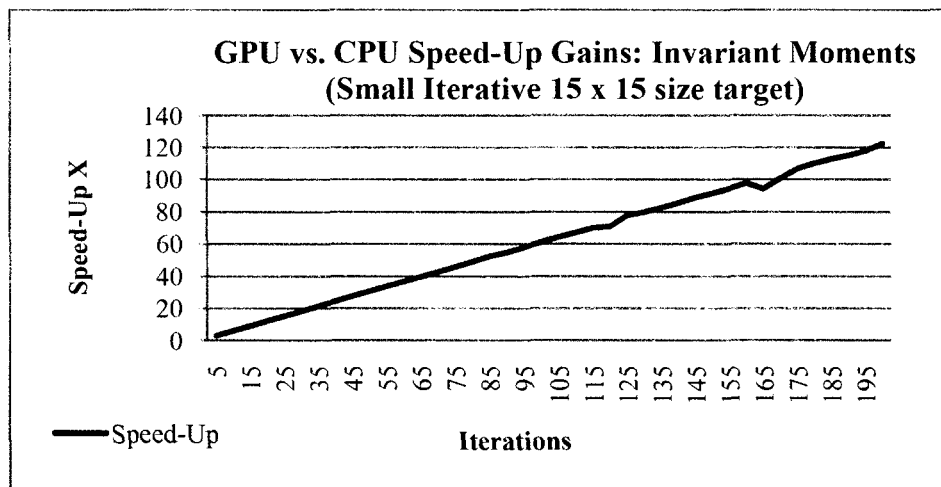


Figure 7.22 GPU-CPU Speed-Up for Invariant Moments (small loops)

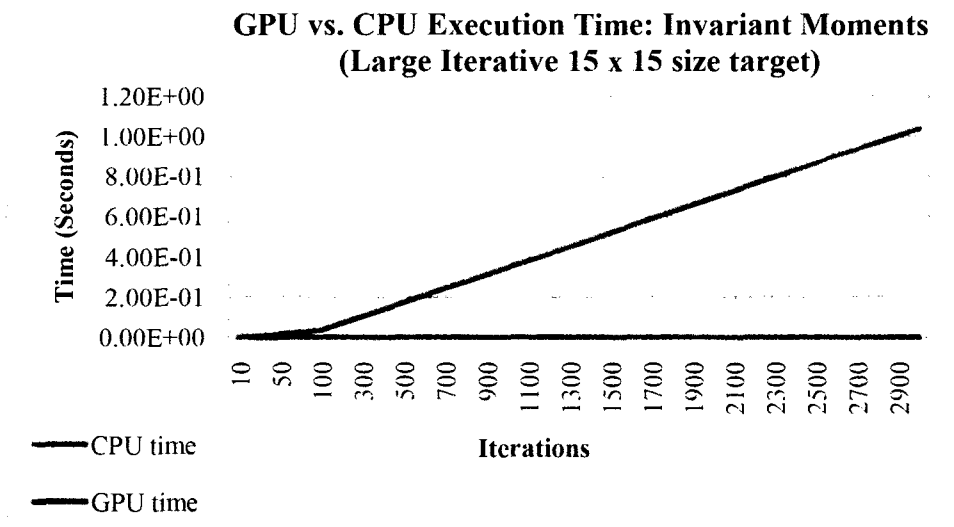


Figure 7.23 GPU-CPU Execution Time for Invariant Moments (large loops)

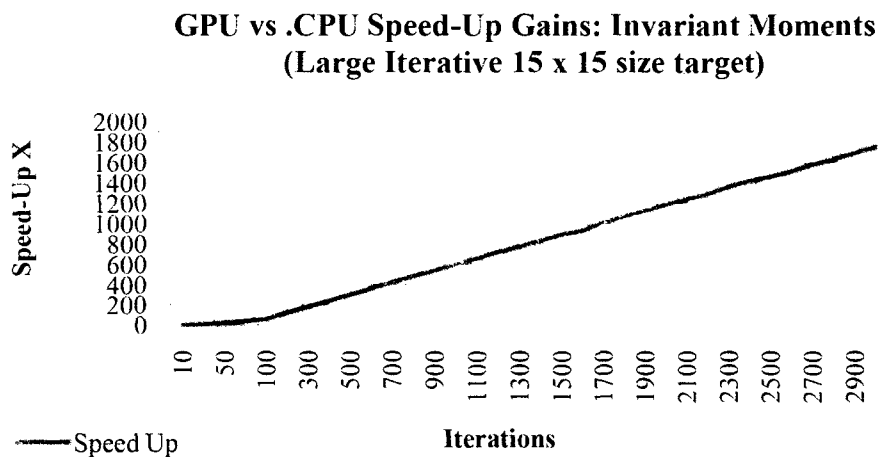


Figure 7.24 GPU-CPU Speed-Up for Invariant Moments (large loops)

7.2.4 GPU for Object Tracking

Described in both Sections 6.2 and 6.3, a correlation-based feature tracker for following detected objects through the image sequence has been used. This stage typically requires a searching-type of algorithm where a match score or probability is

calculated at new potential object location. This type of method can easily be inhibited by the size of the features used for the matching (Figure 6.4), as well as the number of locations to compare. GPUs can provide a means to calculate probabilities based on feature matching simultaneously, thus larger feature sizes and locations can be used to provide real time tracking results. GPUs create performance gains when a significant number of locations are present, which can grow rapidly if multiple orientations of each object are used for comparison (described in Section 5.2.2). The normalized cross correlation (Equation 6.9-10) speed-up using the GPU (Figures 7.25-7.30) is presented. For non-iterative, varying vector size, CPUs outperform the GPU implementation. For iterative implementation, the GPU begins to outperform the CPU after 30 iterations, showing a linear speed-up with an increase in loops size.

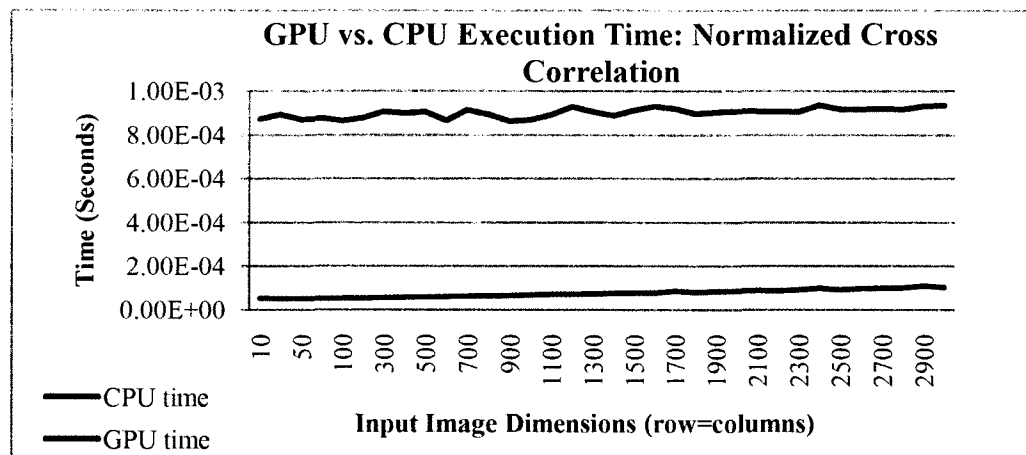


Figure 7.25 GPU-CPU Execution Time for Normalized Cross Correlation (no loops)

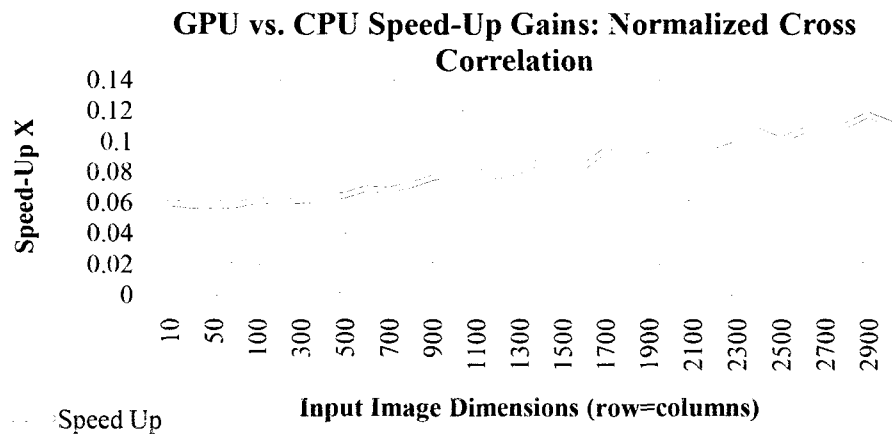


Figure 7.26 GPU-CPU Speed-Up for Normalized Cross Correlation (no loops)

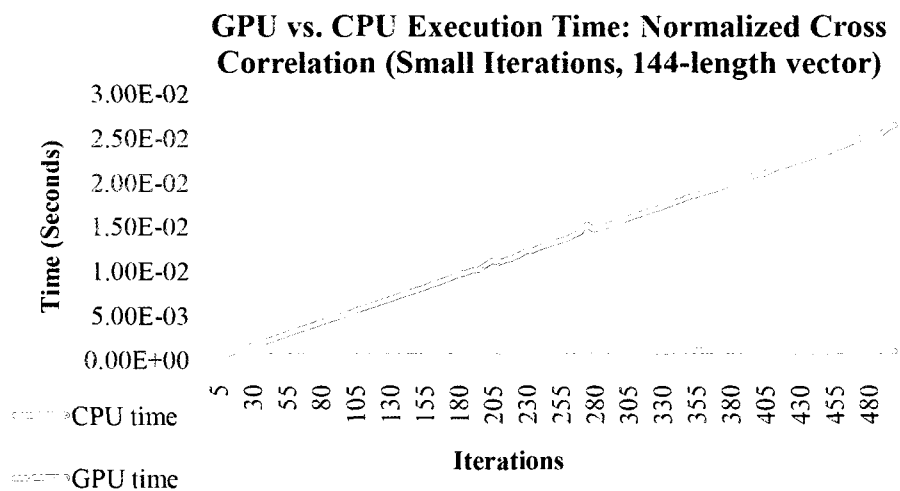


Figure 7.27 GPU-CPU Execution Time for Normalized Cross Correlation (small loops)

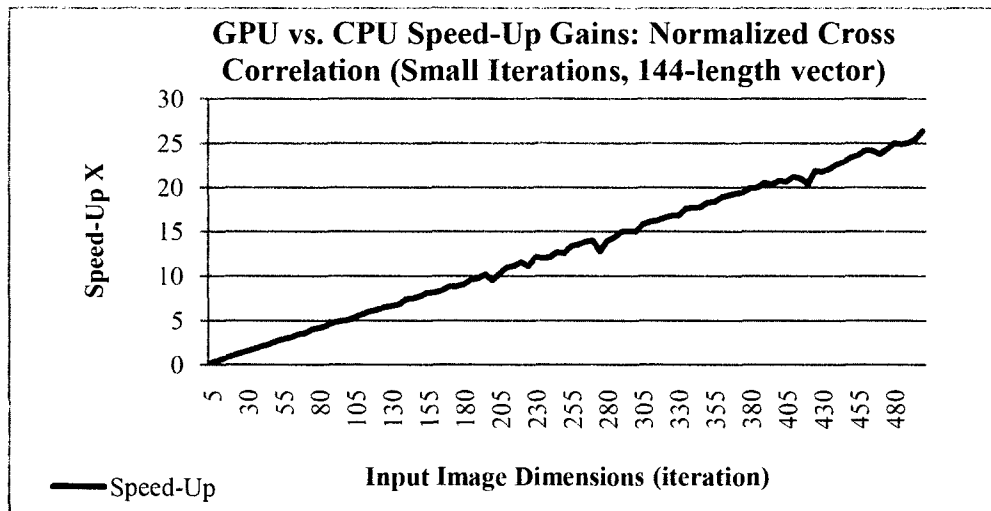


Figure 7.28 GPU-CPU Speed-Up for Normalized Cross Correlation (small loops)

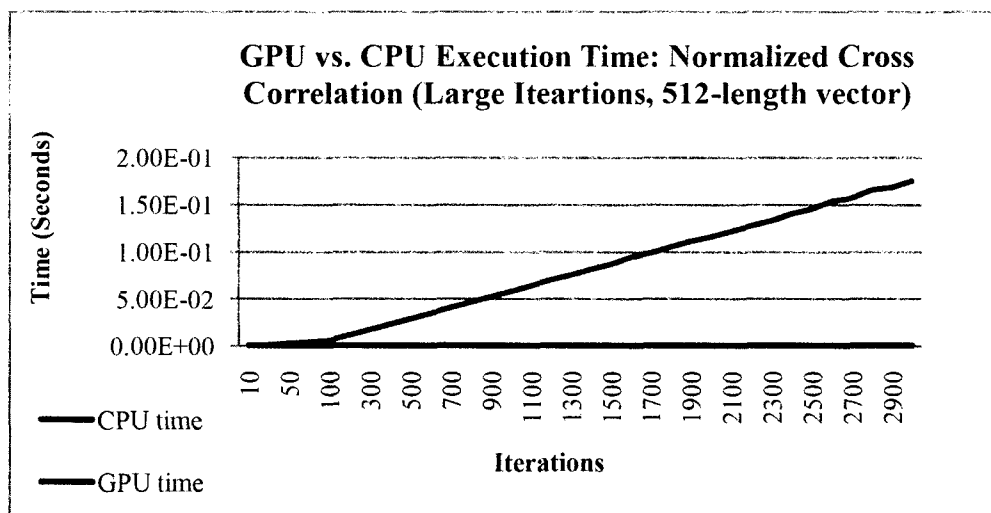


Figure 7.29 GPU-CPU Execution Time for Normalized Cross Correlation (large loops)

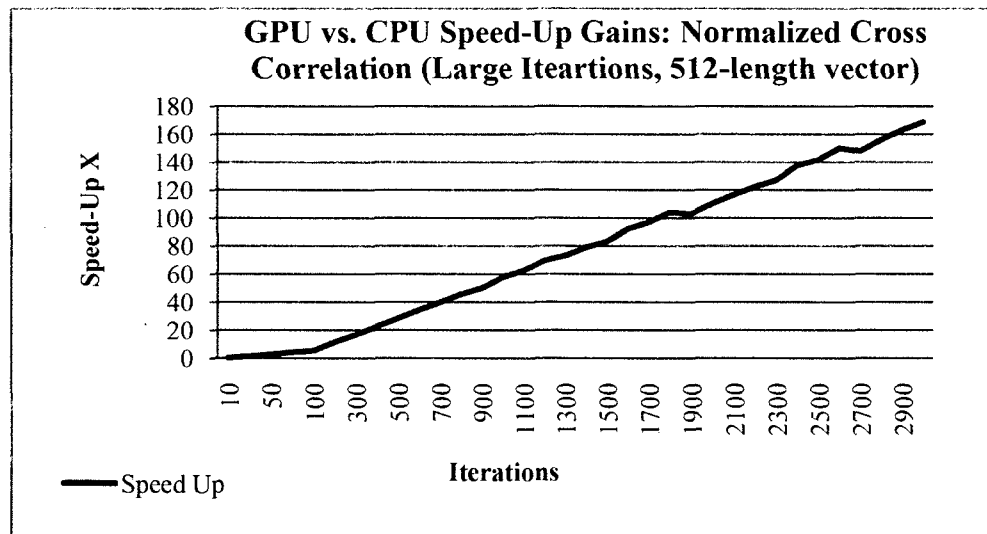


Figure 7.30 GPU-CPU Speed-Up for Normalized Cross Correlation (large loops)

7.2.5 GPU for Implementation Summary

Each of the SE sub-tasks to determine the benefits of GPU-enabled implementations and provided time-gains (in multiples of execution time) have evaluated for each feature calculation in Table 7.1. For non-iterative, highly parallel methods, such as 2-D convolutions (Gaussian filtering) and graph weight initialization, qualitative gains in execution time can be observed. For most feature calculations tested, time lag due to CPU-GPU communication reduces performance to a fraction of the execution time produced by CPU implementations. When feature calculations are required to be performed in an independent iterative manner (as is the case with feature extraction and tracking calculations), large gains are observed at even small iteration sizes (Table 7.2).

Table 7.1 Algorithm GPU Speed-Up for SE

Feature Calculation	Input Size (pixels: row=columns)				
	25	100	250	500	1000
2-D Convolution	0.23	0.80	1.48	2.80	4.06
Graph Weight Initialization	0.25	10.90	288.87	4780.88	14565.00
Graph Norm Cuts	0.02	0.03	0.02	0.07	0.23
Invariant Moments	0.89	0.98	1.00	1.00	1.00
Histogram	0.06	0.05	0.06	0.06	0.07
Normalized Cross Correlation	0.06	0.06	0.06	0.07	0.08

Table 7.2 Iterative GPU Speed-Up for SE

Feature Calculation (For Loop Iterations)	Loop Size				
	25	100	250	500	1000
2-D Convolution	8.90	19.45	77.63	76.85	91.70
Graph Norm Cuts	0.54	2.90	10.51	37.15	156.10
Invariant Moments	15.17	60.93	153.09	300.43	600.77
Histogram	1.25	5.00	12.48	25.00	49.68
Normalized Cross Correlation	1.41	5.94	14.86	30.36	60.32

7.3 GPU-Based SE Framework Conclusion

GPU implementations of SE feature calculations have been tested, and the findings have been discussed. Although GPU provide new real-time methods to SE by leveraging fast, independent feature calculations, each implementation should be analyzed to determine lower bound speed gains. Preprocessing steps that reduce input sizes to increase real-time implementation can potentially be eliminated, which can increase

specificity by leveraging additional object details. At the time of this writing, several limitations still hinder seamless GPU implementation into SE algorithms, such as lack of compatibility for sparse matrices and conditional statements in loops. The independent calculations required to accurately detect, classify, and track objects in increasingly large sensor data will benefit immensely from GPU implementations.

CHAPTER 8

CONCLUSIONS

8.1 Contributions to Object Detection

In Chapter 4, three specific methods for object detection are presented and evaluated which successfully delineate objects from the background in changing, cluttered environments. Although that these methods may not work out-of-the-box for any SE application, each of these methods contribute to the SE community with the hopes that particular methodologies can be used as benchmarks or expanded upon to serve the user's particular needs. The ill-defined nature of object detection will remain a challenging topic for many years to come, with more complex and computationally intensive methods continuously being realized due to the expanding resources available. The research in this field will be continued by leveraging recent investigations with GPU implementation discussed in Chapter 7 with the hopes of using larger input data to increase accuracy. Additionally, more tracking methods will be integrated into the detection stage to further decrease false alarms caused by OCs, with more computationally intensive methods available for real time implementation.

More optical flow and image segmentation methods will be implemented to a create a cascading performance model which can predict algorithm accuracy probabilities based on offline evaluation. This direction can attain an even higher standard of robustness that is not currently found in the object detection literature. Methods such as the one

described in Section 4.4 is only the beginning of how motion and segmentation methods can be integrated into one another for refining detection results.

8.2 Contributions to Feature Extraction and Classification

Several novel feature extraction methodologies that have proven to successfully detect, classify, and track objects in image sequences have been implemented. For detection, the flow and edge gradients have been implemented with a binning methodology, which can help in detecting object with concave boundaries (like objects). Other researchers in persistent surveillance can benefit from these additional features, due to the difficulties in robust segmentation and detection algorithm design. Edge bins also provide useful invariant feature for classification, due to their ability to handle rotation and partial occlusion. Most feature sets cannot handle both of these types of variations in object appearance, which can work well when combined with features that provide high levels of specificity. Edge bins have shown to work well in classifying both multiclass object sets and binary detections by itself, but can still aid in working in a larger group of features. As shown in this dissertation, these features can be sped up by GPUs when used in an independent, iterative manner (which is typically the case).

DRSURF features have provided a novel way to describe a low resolution object with poor second derivatives for tracking. This method will be integrated with other features for a hierarchical tracking methodology, which first uses invariant features to reduce the search space, then performs DRSURF calculations for better tracking results. The DRSURF features can provide other researchers with an additional feature set to benchmark tracking and feature extraction methods.

Multiple methods for supervised classification for both detection and multiclass labeling have also been thoroughly investigated and evaluated. Several methods generated high accuracy, Random Forest performed best in real-time scenarios due to its high accuracy and fast training speed. The Random Forest detection scheme will be tested on other data sets that provide training data to study the robust discrimination strength of the classifier. MLP and KSTAR have also displayed accurate results when trained offline, but real-time classification requirements may inhibit their use.

8.3 Contributions to Object Tracking

Aside from using DRSURF for tracking, the use of tracking methods which can follow objects through rotation, translation, and scaling changes by integrating motion and appearance features have been presented and validated. Although this approach is not unique in terms of combining these two classes of features, the method takes into account specific phenomena that occurs using particular optical flow (phase-based) and gradient features (DRSURF) for an optimal tracking performance. Like object detection problems with robustness, this particular approach cannot provide an out-of-the-box solution for any image sequence that contains objects. Creating more robust tracking is a natural next step for the research in object tracking, with methods that can be incorporated in the detection stage of particular interest. The use of GPUs expands the ability to perform matching on larger candidate spaces for reductions in false negatives, but must be counteracted by high selective tracking methods. Such tracking methods should incorporate a multi-tracking scheme, which can choose a particular tracker's estimation based on tracker performance models. Although the presented tracker does not currently provide this level of complexity, the powerful detection methodology may

play a more significant role in tracking, which is referred as detect-before-track. These methods suffer from higher computational costs and reduced robustness to occlusions, which will be mitigated by implementing GPU versions for speed increases and using dynamic state models.

REFERENCES

- [1] B. Bhanu, "Automatic Object Recognition: State of the Art Survey," *IEEE J. Aerospace and Electronic Systems*, vol. AES-22, no. 4, pp. 364-379, Feb. 2007.
- [2] J. Han and M. Kamber. *Data Mining Concepts and Techniques*. San Francisco, CA: Morgan Kaufman Publishers, 2001.
- [3] M. Motwani, M. Gadiya, R. Motwani, and F. C. Harris, Jr., "A Survey of Image Denoising Techniques," *Proc. of GSPX 2004*, Santa Clara, CA, Sept. 2004.
- [4] R. Yang, L. Yin, M. Gabbouj, J. Astola, and Y. Neuvo, "Optimal Weighted Median Filters under Structural Constraints," *IEEE Journal of Signal Processing*, vol. 43, pp. 591-604, Mar. 1995.
- [5] A. Ben Hamza, P. Luque, J. Martinez, and R. Roman, "Removing Noise and Preserving Details with Relaxed Median Filters," *Journal of Mathematical Imaging and Vision*, vol. 11-2, pp. 161-177, Oct. 1999.
- [6] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1989.
- [7] H. Zhang, Aria Nosratinia, and R. O. Wells, Jr., "Image Denoising via Wavelet-Domain Spatially Adaptive FIR Wiener Filtering," *IEEE Proc Int. Conf. Acoustics, Speech, and Signal Processing*, Istanbul, Turkey, June 2000.
- [8] D. L. Donoho and I. M. Johnstone, "Ideal Spatial Adaption via Wavelet Shrinkage," *Biometrika*, vol. 81, pp. 425-455, Sept. 1994.
- [9] E. P. Simoncelli and E. H. Adelson, "Noise Removal via Bayesian Wavelet Coring," *3rd Int. Conf. Image Processing*, Lausanne, vol. I, pp. 379-382, Sept. 1996.
- [10] R. G. Baraniuk, "Optimal Tree Approximation with Wavelets," *Proc. SPIE Technology Conf. Wavelet Applications Signal Processing VII*, Denver, CO, 1999, vol. 3813, pp. 196-207.

- [11] W. Buccigrossi and E. P. Simoncelli, "Image Compression via Joint Statistical Characterization in the Wavelet Domain," *J. IEEE Image Processing*, vol. 8, no. 12, pp. 1688-1701, Dec. 1999.
- [12] T. D. Bui and G. Y. Chen, "Translation-Invariant Denoising using Multi-Wavelets," *J. IEEE Transactions on Signal Processing*, vol. 46, no. 12, pp. 3414-3420, 1998.
- [13] I. Cohen, S. Raz, and D. Malah, "Translation Invariant Denoising using the Minimum Description Length Criterion," *Signal Processing*, vol. 75, no. 3, pp. 201-223, 1999.
- [14] A. Jung, "An Introduction to a New Data Analysis Tool: Independent Component Analysis," *Proceedings of Workshop GK "Nonlinearity,"* Regensburg, Oct. 2001.
- [15] R. C. Gonzalez and R.E. Wood. *Digital Image Processing*. Reading, MA: Addison Wesley, 2002.
- [16] J. C. Russ, *The Image Processing Handbook: Fourth Edition*, Boca Raton, FL: CRC, 2002.
- [17] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *Philosophical Magazine*, vol. 2, no. 6, pp. 559-572, 1901.
- [18] C. Rader and N. Brenner, "A New principle for Fast Fourier Transformation," *J. IEEE Acoustics, Speech, and Signal Processing*, vol. 24, pp. 264-266, Jan. 2003.
- [19] S. Mallat, *A Wavelet Tour of Signal Processing*, New York, NY: Academic Press, 1999.
- [20] M. Foracchia, E. Grisan, and A. Ruggeri, "Luminosity and Contrast Normalization in Retinal Images," *Medical Image Analysis*, vol. 9, pp. 179-190, June 2005.
- [21] B. K. P. Horn, *Robot Vision*, Cambridge, MA: MIT Press, 1986.
- [22] B. K. P. Horn and B.G. Schunck, "Determining Optical Flow," *Technical Report A.I. Memo 572*, Massachusetts Institute of Technology, 1980.
- [23] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of Optical Flow Techniques," *International Journal of Computer Vision*, vol. 12, pp. 43-77, Feb. 1994.

- [24] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *Proc 7th Int. Joint Conf. on Artificial Intelligence, IJCAI*, pp. 674-679, 1981.
- [25] D. J. Fleet and A. D. Jepson, "Computation of Component Image Velocity from Local Phase Information," *Int. J. Computer Vision*, vol. 5, no. 1, pp. 77-104, 1990.
- [26] J. Canny, "A Computational Approach to Edge Detection," *J. IEEE Pattern Analysis and Machine Intelligence PAMI*, vol. 8, pp. 679-698, 1986.
- [27] E. Saber, A. M. Tekalp, and G. Bozdagi, "Fusion of Color and Edge Information for Improved Segmentation and Edge Linking," *J. Image Vis. Comput.*, vol. 15, no. 1197, pp. 769-780, 1997.
- [28] M. Sezgin and B. Sankur, "Survey Over Image Thresholding Techniques and Quantitative Performance Evaluation," *J. Electronic Imaging*, vol. 13, pp. 146-165, 2004.
- [29] F. Meyer, "Topographic Distance and Watershed Lines," *Signal Processing*, vol. 38, pp. 113-125, 1994.
- [30] C. Sinthanayothin, J. F. Boyce, T. H. Williamson, H. L. Cook, E. Mensah, S. Lal, and D. Usher, "Automated Detection of Diabetic Retinopathy on Digital Fundus Images," *Diabetic Medicine: A Journal of the British Diabetic Association*, vol. 19, pp. 105-112, Feb. 2002.
- [31] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *CVPR '97: Proc. 1997 Conf. on Computer Vision and Pattern Recognition (CVPR '97)*, Washington, DC, USA: IEEE Computer Society, 1997.
- [32] J. Malcolm, Y. Rathi, and A. Tannenbaum, "Graph Cut Segmentation with Nonlinear Shape Priors," in *IEEE Int. Conf. on Image Processing, 2007. ICIP 2007*, vol. 4, 2007.
- [33] N. Vu and B.S. Manjunath, "Shape Prior Segmentation of Multiple Objects with Graph Cuts," *IEEE Intern. Conf. on Computer Vision and Pattern Recognition. (CVPR)*, 2008.
- [34] C. H. Fosgate, H. Krim, W. W. Irving, W. C. Karl, and A. S. Willsky, "Multiscale Segmentation and Anomaly Enhancement of SAR Imagery," *J. IEEE Image Processing*, vol. 6, pp. 7-20, Aug. 2002.
- [35] K. L. Vincken, A. S. E. Koster, and M. A. Viergeever, "Probabilistic Multiscale Image Segmentation," *J. Pattern Analysis and Machine Intelligence*, vol. 19, pp. 109-120, 1997.

- [36] H. Choi and R. G. Baraniuk, "Multiscale Image Segmentation using Wavelet-Domain Hidden Markov Models," *J. Image Processing*, vol. 10, pp. 1309-1321, Aug. 2002.
- [37] J. B. Kim, C. W. Lee, K. M. Lee, T. S. Yun, and H. J. Kim, "Wavelet-Based Object Tracking for Automatic Traffic Surveillance," *IEEE TENCON'01*, vol. 1, pp. 313-316, Singapore, Aug. 2001.
- [38] A. Bugeau and P. Pérez, "Track and Cut: Simultaneous Tracking and Segmentation of Multiple Objects with Graph Cuts," *Journal of Image Video Processing*, vol. 1, pp. 1-14, 2008.
- [39] D. J. Salmond and H. Birch., "A Particle Filter for Track-before-Detect," *Proc. American Control Conference*. 2001. Arlington, VA, USA.
- [40] Y. Boers, H. Driessen, J. Torstensson, M. Trieb, R. Karlsson, and F. Gustafsson, "Track-before-Detect Algorithm for Tracking Extended Objects," *Proc. IEEE on Radar and Sonar Navigation*, vol. 153, pp. 345-351, 2006.
- [41] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt, "Hierarchy and Adaptivity in Segmenting Visual Scenes," *Nature*, vol. 442, pp. 810-813, Aug. 2006.
- [42] K. Mikolajczyk and C. Schmid, "Scale and Affine Invariant Interest Point Detectors," *Int. J. Computer Vision*, vol. 1, no. 60, pp. 63-86, 2004.
- [43] C. Schmid and R. Mohr, "Local Grey-value Invariants for Image Retrieval," *J. IEEE Pattern Analysis and Machine Intelligence*, vol. 19, pp. 530-535, 1997.
- [44] K. Mikolajczyk and C. Schmid, "A Performance Evaluation of Local Descriptors," *J. Pattern Analysis and Machine Intelligence*, vol. 27, pp. 1615-1630, 2005.
- [45] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Computer Vision*, vol. 60, pp. 91-110, Nov. 2004.
- [46] H. Bay, A. Ess, T. Tuytelaars, and L. Vangool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, pp. 346-359, June 2008.
- [47] J. Xiao, H. Cheng, H. Feng, and C. Yang, "Object Tracking and Classification in Aerial Videos," *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, May 2008.

- [48] W. Freeman and E. Adelson, "The Design and Use of Steerable Filters," *J. IEEE Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, pp. 891-906, 1991.
- [49] J.-Y. Choi and Y.-K. Yang, "Object Detection from Aerial Images using Local Shape Information," *Advances in Image and Video Technology*, vol. 5414, ch. 20, pp. 227-236, 2009.
- [50] S. Belongie, J. Malik, and J. Puzicha, "Shape Matching and Object Recognition using Shape Contexts," *J. IEEE Pattern Analysis and Machine Intelligence*, vol. 24, no. 4, pp. 509-522, Apr. 2002.
- [51] D. Cremers, T. Kohlberger, and C. Schnörr, "Shape Statistics in Kernel Space for Variational Image Segmentation," *Pattern Recognition*, vol. 36, no. 9, pp. 1929-1943, Sept. 2003.
- [52] T. Ojala, M. Pietikainen, and T. Maenpää, "Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns," *J. IEEE Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971-987, 2002.
- [53] L. Van Gool, T. Moons, and D. Ungureanu, "Affine / Photometric Invariants for Planar Intensity Patterns," *Proceedings of the 4th European Conference on Computer Vision, Cambridge, UK*, pp. 642-651, 1996.
- [54] A. Ashbrook, N. Thacker, P. Rockett, and C. Brown, "Robust Recognition of Scaled Shapes using Pairwise Geometric Histograms," *Proc. 6th British Machine Vision Conference, Birmingham, UK*, pp. 503-512, 1995.
- [55] G. Dorko and C. Schmid, "Selection of Scale-Invariant Parts for Object Class Recognition," *Proc. of the 9th Int. Conf. Computer Vision, Nice, France*, pp. 634-640, 2003.
- [56] R. M. Haralick, K. Shanmugan, and I. Dinstein, "Textural Features for Image Classification," *J. IEEE Systems, Man, and Cybernetics*, vol. SMC-3, pp. 610-621, 1973.
- [57] D. Gabor, "Theory of Communication," *J. I.E.E.E.*, vol. 3, no. 93, pp. 429-457, 1946.
- [58] J. K. M. Vetterli, *Wavelets and Subband Coding*, Prentice Hall, 1995.
- [59] L. Eikvil, L. Aurdal, and H. Koren, "Classification-Based Object Detection in High-Resolution Satellite Images," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 64, pp. 65-72, Jan. 2009.

- [60] L. Cao, "Classification SAR Objects with Support Vector Machine," *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, Feb. 2007.
- [61] A. Sadjadi and X. Yu, "Neural Network Directed Bayes Decision Rule for Moving Target Classification," *IEEE Trans. On Aerospace and Electronic Systems*, vol. 36, pp. 176-188, Jan. 2000.
- [62] A. Yilmaz, O. Javed, and M. Shah, "Object Tracking: A Survey," *ACM Comput. Surv.*, vol. 38, no. 4, pp. 13-20, 2006.
- [63] S. Avidan, "Support Vector Tracking," *J. IEEE Pattern Analysis and Machine Intelligence*, vol. 26, no. 8, pp. 1064-1072, Aug. 2004.
- [64] O. Williams, A. Blake, and R. Cipolla, "Sparse Bayesian Learning for Efficient Visual Tracking," *J. IEEE Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1292-1304, 2005.
- [65] S. Park, J. K. Aggarwal, "A Hierarchical Bayesian Network for Event Recognition of Human Actions and Interactions," *Multimedia Systems*, vol. 10, no. 2, pp. 164-179, 2004.
- [66] S. J. Julier and J.K. Uhlmann, "A New Extension of the Kalman Filter to Nonlinear Systems." In *Proc. SPIE - Int. Soc. Opt. Eng. (USA)* (Orlando, FL, Apr. 1997), vol. 3068, pp. 182-193.
- [67] Y. Bar-Shalom, T. Foreman, *Tracking and Data Association*. Academic Press Inc., 1988.
- [68] H. Tanizaki, "Non-Gaussian State-Space Modeling of Nonstationary Time Series," *J. American Statistics Association*, vol. 82, pp. 1032-1063, 1987.
- [69] Y. Li, H. Ai, T. Yamashita, S. Lao, and M. Kawade, "Tracking in Low Frame Rate Video: A Cascade Particle Filter with Discriminative Observers of Different Life Spans," *J. IEEE Pattern Analysis and Machine Intelligence*, vol. 30, no. 10, pp. 1728-1740, Oct. 2008.
- [70] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," Carnegie Mellon University, Tech. Rep. CMU-CS-91-132, April 1991.
- [71] D. Comaniciu and P. Meer, "Mean Shift: A Robust Approach toward Feature Space Analysis," *J. Pattern Analysis and Machine*, vol. 24, no. 5, pp. 603-619, Aug. 2002.
- [72] A. Bugeau and P. Pérez, "Track and Cut: Simultaneous Tracking and Segmentation of Multiple Objects with Graph Cuts," *J. Image Video Process.*, vol. 2008, pp. 1-14, 2008.

- [73] J. S. Lim, *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, p. 548, equations 9.44-9.46, 1990.
- [74] L. F. Chen, H. Y. M. Liao, and J. C. Lin, "Wavelet-Based Optical Flow Estimation," *J. IEEE Circuits Syst. Video Technol.*, vol. 12, no. 1, pp. 1-12, Feb. 2002.
- [75] R. G. Baraniuk, "Optimal Tree Approximation with Wavelets," *Proc. SPIE Tech. Conf. Wavelet Applications Signal Processing VII*, vol. 3813, pp. 196-207, Denver, CO, 1999.
- [76] J. Bouguet, "Pyramidal Implementation of the Lucas-Kanade Feature Tracker: Description of the Algorithm," Technical report, OpenCV Document, Intel Microprocessor Research Labs, 2000.
- [77] T. Gautama and M. A. Van Hulle, "A Phase-Based Approach to the Estimation of the Optical Flow Field using Spatial Filtering," *J. IEEE Neural Networks*, vol. 13, pp. 1127-1136, 2002.
- [78] M. M. Van Hulle, "A Goal Programming Network for Linear Programming," *Biol. Cybernetics*, vol. 65, pp. 243-252, 1991.
- [79] Columbus Large Image Format (CLIF) Dataset. [Online]. Available: <https://www.sdms.afrl.af.mil/datasets/clif2007/>
- [80] E. Sharon, A. Brandt, and R. Basri, "Fast Multiscale Image Segmentation," *J. IEEE Computer Vision and Pattern Recognition*, vol. 1, pp. 1070-77, 2000.
- [81] M. K. Hu, "Visual Pattern Recognition by Moment Invariants," *IRE Trans. Info. Theory*, vol. IT-8, pp. 179-187, 1962.
- [82] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, June 2005.
- [83] J. G. Cleary and L. E. Trigg, "K*: An Instance-Based Learner Using an Entropic Distance Measure," *Proc. of the 12th Int. Conf. on Machine Learning*, pp. 108-114, 1995.
- [84] J. Fung and S. Mann, "OpenVIDIA: Parallel GPU Computer Vision," *ACM MULTIMEDIA*, pp. 849-852, 2005.

- [85] Y. Allusse, P. Horain, A. Agarwal, and C. Saipriyadarshan, "GPUCV: A GPU-Accelerated Framework for Image Processing and Computer Vision," *Proc. of the 4th Int. Sym. Advances in Visual Computing, Part II*, pp. 430-439, Springer, 2008.
- [86] NVIDIA Corporation. NVIDIA CUDA Programming Guide, version 1.1, 2007.
- [87] T. Larsen, Aalborg University, Denmark, Feb., 2010. [Online]. Available. http://wiki.accelereyes.com/wiki/index.php/GPU_Memory_Transfer.