ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF TECHNOLOGY

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

AUTOMATIC SEMANTIC VIDEO OBJECT SEGMENTATION

By

Eneyachew Tamir

A thesis submitted to the school of Graduate studies of Addis Ababa University in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Engineering

December 2007

Addis Ababa, Ethiopia

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

AUTOMATIC SEMANTIC VIDEO OBJECT SEGMENTATION

By

Eneyachew Tamir

Advisor

Dr Kumudha Raimond

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

AUTOMATIC SEMANTIC VIDEO OBJECT SEGMENTATION

By

Eneyachew Tamir

FACULTY OF TECHNOLOGY

APPROVAL BY BOARD OF EXAMINERS

Dr. Mengesha Mamo	_____
Chairman Dept. of Graduate Committee	Signature
Dr. Kumudha Raimond	_____
Advisor	Signature
_____	_____
Internal Examiner	Signature
_____	_____
External Examiner	Signature

ACKNOWLEDGEMENT

First of all I thank The Almighty God for letting me finish this thesis. The painstaking effort exerted on this thesis would not have been successful without His help.

I would like to express my appreciation to my Supervisor Dr. Kumudha Raimond for her help, starting from providing the research idea, and the overall support and guidance, suggestions and encouragement throughout the course of the work.

I am grateful to all my friends and other people who have been around me for encouraging to successfully finish this thesis. Special thanks go to my brother Mossie who has been living with me in my stay in Addis for the study, without whom things would have been so different.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ACRONYMS

AS	Automatic Segmentation
BMA	Block Matching Algorithm
CD	Change Detection
CIF	Common Interchange Format (video format with size 288x352)
GMEC	Global Motion Estimation and Compensation
GMOB	Global Motion of Background
ISO	International Standards Organization
LMS	Least Median of Squares
LS	Logarithmic Search
MAD	Mean Absolute Difference
ME	Motion Estimation
MI	Motion Information
MO	Moving Object
MPEG	Moving Pictures Experts Group
PP	Post-Processing
RF	Reference Frame
SIF	Source Input Format (video format with size 240x352)
SS	Spatial Segmentation
ST	Spatio-Temporal
SVO	Semantic Video Object
TSS	Three Step Search
VOS	Video Object Segmentation

ABSTRACT

Content-based video processing is one of the methods considered to meet the demands of newly emerging multimedia applications. For content-based processing, video has to be segmented into meaningful objects – semantic video objects. Many applications require automatic segmentation (AS) of semantic video objects. However, AS is very challenging task.

In this thesis, an effective AS system is proposed by examining, selecting and combining efficient and simplified techniques that are justified with theoretical analysis. The proposed system is developed and tested with the following three cases depending on whether there exists camera motion or not in video sequences, and whether there is an initial background reference frame. Case-1 is for video sequences with no camera motion and with initial background reference frame, Case-2 is for video sequences where there is no camera motion and no initial background reference frame, and Case-3 is for video sequences with camera motion.

Change detection is used as the main step in each of the cases to detect semantic objects and to produce object mask. Different problems in change detection like “uncovered background”, “global motion of background (GMOB)” and “camera noise” are identified and solved. Two change detection results are combined to remove the uncovered background problem. More emphasis is made for the problem of GMOB. A 3-level block-based hierarchical motion estimation and affine parameter model for frame warping is used to solve this problem. Camera noise is removed by using model-based change detection.

For post-processing to improve the resulting change detection masks, a new filling-in technique is proposed. This technique is used to fill open areas inside object regions with uniform intensity. To improve the boundary of segmentation masks, morphological open close operations are used. The final semantic video objects are obtained by superimposing the resulting mask over the original frame.

Test results show that the system effectively identified and segmented the semantic objects. Subjective evaluation of results for the three cases showed that among the window sizes used in change detection, 5x5 and 7x7 produced better and comparable results in terms of visual quality and boundary smoothness. These results are obtained after applying one pair of open close operation in Case-1 and two pairs in Case-2 and Case-3.

Based on subjective comparison of results with other systems, 80% of observations for the results of Case-1 and 100% for Case-2 reported more pleasing results with smooth boundary.

Keywords: content-based video processing, automatic segmentation, semantic video objects, change detection, motion estimation, post-processing.

Chapter 1

INTRODUCTION

1.1 Background

The advent of digital technologies and development of computer processing has made producing digital images and video increasingly easier and so there has been a large increase in the amount of visual information in the last decades. This was accompanied by newly emerging multimedia applications. Some of these applications include object-based coding, video databases, interactive video, video conferencing, video surveillance and home video editing. These newly emerging applications and the large increase in amount of visual material raised the need for new methods of processing the visual information that will fulfill the demands of these applications. Some of the demands of these applications are flexible manipulation, reuse of visual contents, efficient storage and fast transmission. To fulfill these demands, video processing has moved from the traditional frame-based processing to content-based processing. The traditional frame-based processing, which supports frame-by-frame coding, provides only limited capabilities in terms of access, manipulation and interaction with visual content. On the other hand, content-based processing provides more general and powerful way of representing and coding content [5].

Video clips are mostly found frame-based. For content-based processing of the video signals, the video data has to be presented based on its contents. The challenge is how to exploit visual information for better use (in coding, storage and content creation). Raw video data is usually in the form of binary streams that are not well organized. For content representation, the raw video data must be decomposed into objects, each object representing particular meaningful content of video. The object-based technique of representing video is gaining

importance in fulfilling the demands of content-based applications stated above. International standards, such as MPEG-4 [37] and MPEG-7 [29], also support object-based representation to provide content-based functionalities. Some of these functionalities include efficient coding, better ways to represent, integrate and exchange the visual information. In fact, it is the MPEG-4 coding standard that introduced the concept of video objects. In MPEG-4 object-based representation, scenes are treated as compositions of audio-visual objects which are separately encoded and decoded. So, identification and separation of meaningful entities or objects from visual data is required as the first step in content-based applications. Meaningful entities are S*emantic Video Objects* (SVO) that correspond to physical objects in real world. The process of separating the meaningful video objects from the background in the video sequence is video object segmentation. Although content-based applications require that video contents should be represented by objects, isolation of objects from video data is left open to researchers. Unfortunately, segmentation of SVOs is not an easy task because it involves complex mechanisms, and it is inherently an *ill-posed* problem [5]. Also, there is no unique definition of SVOs that fits for all applications, i.e. semantics of a particular object is always application dependent [47]. For the success of content-based applications, it is very important that there should be an effective and flexible SVO segmentation system. Although it has been studied for a number of years, SVO segmentation is still considered as one of the most challenging video processing tasks, and therefore it is no doubt that SVO segmentation requires intensive research.

Two strategies are common in semantic VOS research community: semi-automatic [16], [27],[41], in which some kind of user intervention is required to define the semantic object; and automatic[7],[22],[34],[48],[52], where segmentation is performed without user intervention, but usually with some a priori information. Automatic segmentation of SVOs is required by most applications, especially those with real time requirements.

A number of VOS algorithms have been proposed, most aiming to specific applications, and trying to fulfill specific requirements. Promising results have been obtained so far in semi-automatic methods, since there is also human assistance in the segmentation process. However, the human assistance involved in these methods is not required because it adds

burden to users and also it is not suitable for some applications. On the other hand, fully AS systems are still a challenge, although they are required by many applications.

Many AS systems are designed for specific problems and with simplified assumptions like videos with fixed background. So it is necessary to have flexible AS systems for different types of videos. Most of the existing AS systems involve complex techniques. Also each stage of the segmentation process involves computationally intense operations to obtain good segmentation results Thus, reducing the complexity of the techniques involved is required while keeping accuracy of segmentation results. This can be done by selecting efficient algorithms with reduced computational intensity in each step of the segmentation process. Accuracy of segmentation can be improved by applying post-processing.

1.2 General Objective

Given a sequence of raw video frames, the general objective is to develop a flexible system to segment SVOs automatically from the background. The SVOs are defined as any moving objects in the video.

1.3 Specific objectives

Specifically, this thesis aims at:

- Examining and selecting efficient and simplified algorithms in each step of the segmentation process.
- Designing and implementing effective automatic SVO segmentation system for both static and moving background sequences using the algorithms selected.
- Improving the visual quality of segmentation results by applying effective post-processing.

1.4 Scope of the Thesis

The main focus is dealing with problems of fully automatic segmentation of moving objects in video sequences. Videos containing static background and with GMOB are addressed. The problem of global motion is dealt with considering the problem as motion due to camera movement. Different parametric motion estimation techniques are explained. A motion detection method, the change detection, is explained for segmenting moving objects. Especially, statistical methods of change detection are given more attention. The change detection algorithm can work for both fixed and moving camera sequences. Since the output of change detection is a rough object mask with noise the tasks of object mask refinement are also handled. Post-processing strategies for object mask refinement are explained. For the case of multiple video objects in a scene, a single object is defined as any independent component in the object plane after post-processing. The proposed system is validated by implementation and testing.

1.5 Methodology

The method of research used in this thesis is theoretical analysis and assessment of literature. Different existing systems are studied to find out how the systems approached to solve the problem. The different techniques and the mechanisms used in the existing systems are investigated to select efficient techniques to be used in designing the proposed system. Selection of the efficient techniques is made by analyzing and comparing the simplicity and computation complexity involved in the techniques to perform the intended task. Implementation and testing of the design is done to validate the proposed system. Standard test video sequence data of YUV4:2:0 format in CIF and SIF resolution obtained from the Internet are used for testing. To evaluate the results of the proposed system, subjective evaluation technique which is based on visual observation of the segmentation results is used. In addition to our subjective evaluation, a set of five postgraduate students of Electrical and Computer Engineering (Addis Ababa University) are made to observe, compare and judge the results. The results are evaluated based on their visual quality. To view the raw video

sequences as well as segmentation results, the YUV File Player software from the University of Bath [56] is used.

1.6 Outline of the Thesis

This thesis is organized into seven chapters. The first chapter is the introductory chapter. In chapter 2, a review of existing automatic video object segmentation methods is presented. Chapter 3 explains the problem of global motion estimation and compensation, which is part of video object segmentation for moving camera video sequences. The chapter starts with explaining the notion of motion in video and mathematical background of camera motion modeling. It also discusses different camera motion models. In addition different algorithms for block-based motion estimation and compensation are also explained and compared.

In chapter 4, an algorithm for automatic video object segmentation which is used in this thesis, change detection, is presented. This chapter also provides a review of video object segmentation approaches based on change detection. General model of change detection system and the problem of adaptive threshold computation are described. Methods of post-processing are also presented in this chapter.

Chapter 5 presents design of the proposed semantic video object segmentation system. Different parts of the design and the different cases considered in the design are also explained. Selected algorithms for each block of the proposed system and justification of why each algorithm is selected, is presented in this chapter.

Chapter 6 contains implementation and test results of the proposed system. Test results for the different cases are analyzed and discussions are made on the results. Subjective evaluations of the proposed system based on the test results are also contained in this chapter. Chapter 7 summarizes and concludes the work done and presents the possible future works.

Chapter 2

AUTOMATIC VIDEO OBJECT SEGMENTATION: A REVIEW

Video segmentation started with low-level segmentation by partitioning images into regions of similar attributes like color, texture, motion. The obtained regions consist of primitive regions that usually do not have a one-to-one correspondence with physical objects, which may contain regions of totally different homogeneities. However, the ultimate goal of segmentation process is to divide an image into physical objects composing it, so that each region constitutes a semantically meaningful entity (high-level segmentation) [50]. The semantic concept is difficult to formulate based on homogeneity criteria of primitive regions. The semantic (meaning) is defined through a human abstraction and in general human intervention is needed (e.g. a user may define boundary of objects by drawing a line) to identify and segment the meaningful objects, leading to semi-automatic segmentation. However, if the meaningful objects are defined as moving objects, they can be segmented without user intervention. The methods of segmentation which does not require any user intervention are called automatic methods. These methods exploit motion information (MI) present in video to segment SVOs. The motion of a moving object is usually different from the motion of the background and other objects and this can be used to automatically segment moving objects (MOs). The difference in motion between MOs and other parts can be automatically detected by a temporal intensity change between frames in the video sequence.

Several different approaches to segmentation have been proposed in literature for broad varieties of applications. Classifications of segmentation vary significantly in the literature and no consistent classification can be found, in both automatic and semi-automatic domains. However, focus is given on automatic methods, and for the purpose of discussion, automatic methods are roughly classified into two groups: spatio-temporal (ST) and change detection (CD) based. In the following two sections, existing AS methods are discussed based on these groups. Detailed classifications can be found in [54]

2.1 Spatio-temporal (ST) based methods

These methods attempt to exploit and integrate spatial and temporal features inherent in video. It is assumed that GMOB segmentation leads to accurate segmentation results, but with an increased complexity. Temporal segmentation can identify MOs since most MOs have distinct motion patterns from the background. Spatial segmentation (SS) can determine object boundaries accurately if underlying objects have a different visual appearance (color, luminance, etc) from background.

In [17], color and motion features are combined for foreground and background separation. GMOB is done according to color similarity using a non-parametric gradient-based iterative color clustering algorithm called the mean shift algorithm. Moving regions are then identified by a motion detection method, which is developed based on frame intensity difference. After dividing image frame into homogeneous spatial regions, whether each region belongs to foreground/background is determined by motion detection using the 6 parameter region-based affine motion model. In this model, only the moving regions are analyzed after detecting the moving regions. The final object boundaries are post-processed to smoothen edges and eliminate small regions, using morphological operators called open and close (open to remove edge artifacts (unwanted edges), close to fill holes and remove edges). This algorithm produces good results but the iterative color clustering for region segmentation may not converge easily, and increases computation intensity. Also, this approach deals only for MOs with still background, and cannot be applied for video sequences with moving background.

The authors in [6] used motion as clue to semantic information. In their framework, an automatic partition based on color change detection is presented. Homogeneous regions are detected using the fuzzy C-means multi-feature (color and texture) clustering approach. The grouping of regions into objects is driven by semantic interpretation of the scene, which depends on the specific application at hand. This approach works for classes of applications where meaningful objects are MOs. The semantic partition is automatically computed using

color change detection. They assumed that the camera is fixed and this assumption is the one that limits the system from being generic.

The paper in [20] presents automatic segmentation of MOs in natural video sequences. This approach utilizes global motion estimation and compensation in combination with spatial image segmentation and segment-based diffusion of local MI. Global motion estimation is applied to generate a large background image, by spatially integrating a number of previous and subsequent frames. Video objects moving differently in comparison with global motion of the background are identified as deviations in the background image. After spatial image segmentation, motion feature is diffused through image segments, which also allows identifying still parts of the video objects which cannot be detected using other techniques. After GMOB diffusion of the motion feature, each image segment is classified as being in the foreground or in the background of the scene, which after region merging leads to VOS masks. This method provides a more generic segmentation by considering motion of background. However, the GMOB as well as integration of a number of frames complicates the segmentation process. Since a number of subsequent (future) frames are integrated, there will be a lot of delay introduced.

The authors in [49] formulate the problem of automatic VOS as graph labeling approach over a region adjacency graph based on MI, that is, the approach is to classify regions obtained in an initial partition as foreground or background, based on MI. An initial spatial partition of each frame is obtained by the watershed algorithm [38]. The motion of each region is estimated by hierarchical region matching. A classification stage labels regions as foreground or background. It begins with an initial classification based on statistical significance test which marks regions as foreground candidates.

A clearly visible problem in GMOB approaches is that the GMOB process increases the complexity of the segmentation system, since it is a computationally intensive process, and involves a number of steps. GMOB usually produces over-segmentation of frames, and region merging is required to reduce the number of regions. Applying motion estimation (ME) for each region during classification into foreground and background is a computation

intensive process, which is another limitation of these methods. Also, most of the methods in this class do not consider motion of background (simplified assumptions of videos with fixed backgrounds). So if global motion is to be considered to make them more generic, the complexity will escalate. Hence turning to methods with reduced complexity of algorithms involved is necessary.

2.2 Automatic segmentation (AS) based on change detection

Change detection is a widely used method of motion detection algorithm due to its simplicity and efficiency. It detects changes between two frames caused by movement of objects observed at two different times. Hence, it is used for detecting MOs efficiently and also is ideal for automatic object detection and segmentation. For video sequences with static backgrounds, change detection effectively compares the current frame with the background frame. When the background frame is difficult to obtain, change detection compares the current and the previous frame. In this situation however, one problem arises, which is detection of uncovered background as change. Automatic VOS based on change detection is reported to be more efficient than the GMOB approaches because it is only the motion feature that is used to distinguish the (moving) object from the background. Algorithms that perform GMOB at first will waste much of the computing power in segmenting the background also without knowing the MI [10].

Background subtraction is the most frequently solved problem in MO segmentation by applying change detection. In this scheme, the basic step is the selection of GMOB (RF) or background to be subtracted. In some approaches, accumulated frame difference pictures are analyzed to reconstruct stationary scene component to compare with frames to detect change. In these approaches, there is a strong assumption of stationary background. Other approaches align consecutive frames to construct a reference background image before applying change detection. Whenever there is a big deviation of the background, it is updated as necessary. Changes between two frames can also be identified by using two consecutive frames instead of using a reference background frame.

The algorithm proposed in [7] is a VOS algorithm based on change detection and background updating that can obtain video object from video sequence. First, input frames are Gaussian-smoothed to reduce the effect of noise. Change detection is used to analyze temporal information between successive frames more efficiently than ME. Uncovered background and still object are inherent problems in change detection. To solve these problems, and to acquire initial object mask, frame difference mask and background subtraction masks are combined in this algorithm. The algorithm is dedicated to separate moving object regions from other parts of the scene by using MI. The idea is to reduce computational complexity using change detection. Accuracy of segmentation results is improved through boundary refinement. However, the background generation/ updating, increases computational intensity of the algorithm.

A technique which addresses problem of moving camera (which is often encountered in real life for target tracking surveillance, etc.) is presented in [48]. This object detection algorithm uses three consecutive video frames: backward frame, frame of interest and forward frame to solve the uncovered background. First, optical flow based simultaneous iterative camera motion compensation and background estimation is carried out on backward and forward frames. Differences between camera motion compensated backward and forward frames with the frame of interest are then tested against the estimated background models for intensity change detection. Estimate of the background is generated during the compensation process. Next, these change detection results are combined together for acquiring approximate shape of the MO. The moving area information is then merged with region information in terms of region boundary to obtain the final result. This approach effectively solved the problem of uncovered background. However, the iterative optical flow based method used for camera motion compensation is computationally intensive [47]. Also, estimating and generating background model complicates the segmentation process.

Although approaches based on change detection discussed above try to simplify AS, as compared to GMOB approaches, they also have their own problems. Specifically, the complexity of background generation and updating requires more attention in these methods. If there is a static reference background frame, the problem will be easier and accurate results

may be obtained. However, when the background itself is in motion, and when no initial background reference (a frame prior to appearance of any moving foreground object in the video sequence) is present, the problem will be more complex, and the results of segmentation my not be accurate. This shows that there is still a lot to be done to obtain better segmentation system.

In this thesis, change detection is used as the main mechanism for automatic segmentation, and for reduced number of steps and computation intensity involved in the segmentation process. Video sequences containing both static and moving backgrounds are considered to make the system more flexible. Although change detection provides these advantages, there are also problems when using change detection in automatic segmentation that need to be solved. These include global motion of background, uncovered backgrounds, camera noise and variations in scene illumination. Especially, the problems of global motion of background and uncovered backgrounds are very serious and are given more attention. For camera in motion video, background motion is compensated first, before change detection is applied. To solve the problem of uncovered background (Figure 2.1) for video sequences whose background frame is difficult to obtain, two change detection results are combined.

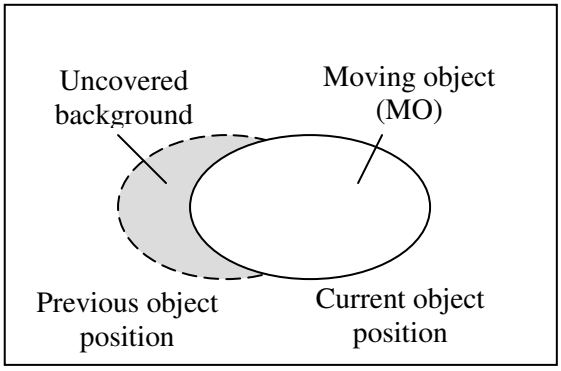


Figure 2.1 Problem of uncovered background. When the MO changes its position from previous to current position, the uncovered background will be revealed.

The complete system proposed is presented in chapter 5. The next chapter discusses camera motion estimation and compensation methods and chapter 4 presents discussion on change detection algorithms. These two chapters (chapter 3 and 4) also examine and compare the different mechanisms involved in each of their respective processes.

Chapter 3

GLOBAL MOTION ESTIMATION AND COMPENSATION

3.1 The notion of motion in video

Video has the concept of motion which is a very useful feature for discrimination of MOs. Two kinds of motion can be identified [5]: 2-D motion and apparent motion. 2-D motion is the projection of the 3-D real world motion into the image plane. The 2-D motion from one image plane to another can be described by *motion vectors*, which are the projections of *displacement vectors* in 3-D space. Figure 3.1 shows projection of a MO on to an image plane, and the relation between 3-D and 2-D motion. When an object $P(x,y,z)$ at time t_0 in 3-D space is moved to $P(x',y',z')$ at time $t_1 = t_0 + dt$, its projected image in 2-D plane (camera plane) moves from $P(x,y)$ to $P(x',y')$, where $(x',y') = (x+dx, y+dy)$. The 3-D motion vector at $P(x,y,z)$ is the 3-D displacement $D(P,t_0,t_1) = P(x,y,z) - P(x',y',z')$ and the 2-D displacement $d(P,t_0,t_1) = P(x,y) - P(x',y')$.

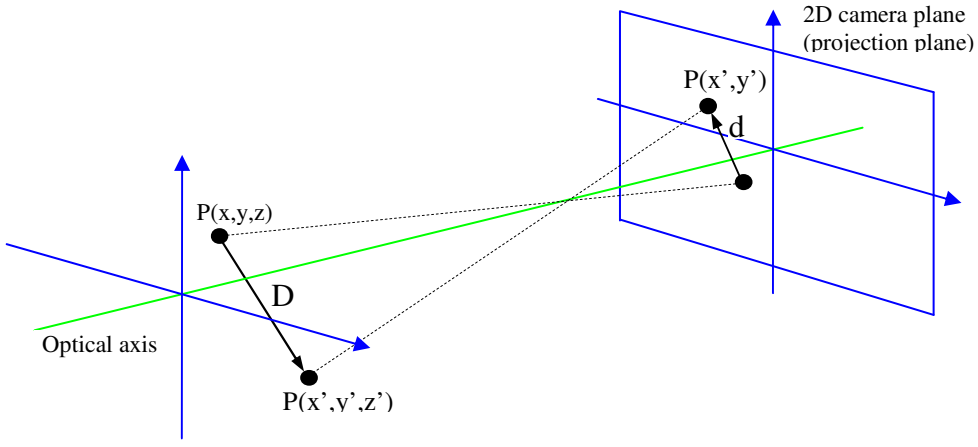


Figure 3.1 Projection of a moving point on to camera plane

The distribution of such motion vectors over the image plane constitutes the *motion field*. The motion field of points on the image plane at time t_1 is represented by $d(P,t_0,t_1)$. Apparent motion is the motion that is considered when analyzing temporal changes. In fact, motion cannot be directly measured in image sequences. The only available observation that can be measured is the time varying intensity function. So, only the apparent motion can be computed, as an estimate of the 3-D motion.

Motion estimation (ME) aims at characterizing this apparent motion by motion field or by the temporal variation of intensities, which is termed as optical flow field. The fundamental assumption for ME is that the luminance/color I of a pixel $P(x, y)$ on MOs remains constant along P' s motion trajectory [47]:

$$I(x(t), y(t), t) = C \tag{3.1}$$

This is called the optical flow constraint. So, ME is closely related to analyzing luminance/color value changes in spatial and temporal domain. The motion of pixels or regions is determined by identifying the position of the corresponding regions/pixels in successive frames. Corresponding pixel/region is determined by searching and minimizing a criterion. The motion field can be estimated by parametric or non-parametric models. In non-parametric motion field, a dense field is estimated, where a motion field is assigned to each pixel. In parametric motion field estimation, the scene is divided into regions and the motion of each region is described by a set of parameters. Different motion models for parametric motion estimation exist in literature, and are discussed in section 3.2.

Motion in video sequences is caused by two different sources: local and global. Local motion, or object motion, is movement of group of some pixels in the image sequence which is the projection of independent moving real objects on the image frames. Global motion is motion of the whole or majority of the frame. It is usually considered as motion due to movement of camera [3].

When the camera is stationary, the problem of MO segmentation becomes identifying the set of pixels which represent objects from a scene in stationary background. In camera in motion sequences, the task of identifying video objects is difficult, since object motions are 'disturbed' by camera motion [47]. This undesired motion should be first removed before segmentation of the MO is done. While general non-rigid object motion is difficult to describe, camera motion can be modeled with a small set of parameters since the variety of possible camera motions is limited [14]. In the following sections, the different types of camera motions and different motion models for motion estimation are discussed. Also, the methods of motion field estimation, especially the block-based methods are reviewed, giving more attention to these methods due to their simplicity. In the last section of this chapter, the methods of motion parameter estimation and how global motion is compensated are explained.

3.2 Camera motion modeling

Camera motion produces global motion field across the whole image. The different types of camera motions are explained below (Figure 3.2):

Camera motions:

- Track: Horizontal translation.
- Boom: Vertical translation.
- Dolly: Translation in the direction of the optical camera axis.
- Pan: Turning around the vertical axis of the camera.
- Tilt: Turning around the horizontal axis of the camera.
- Roll: Rotation around the optical axis of the camera.
- Zoom: The camera changes its focal length

In vertical or horizontal movements, motion vectors are approximately parallel and magnitudes of motion vectors are approximately the same. In case of zooming, the fields of motion vectors have focus of expansion (zoom in) and focus of contraction (zoom out) (Figure 3.3).

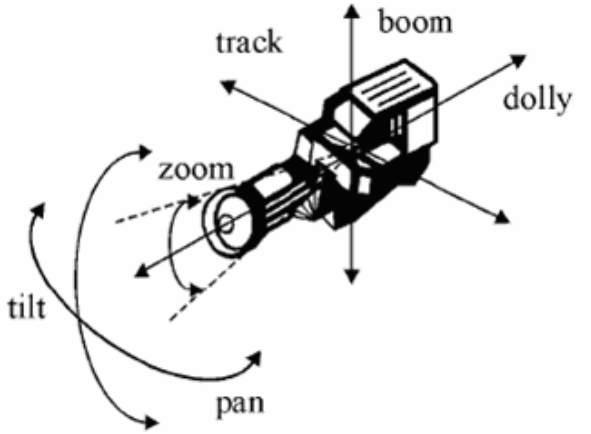


Figure 3.2 Camera motion operations

Global motion or camera motion can be removed in order to simplify further object segmentation. Changes in sequence of frames due to camera motion are modeled and estimated, relative to some GMOB (RF). The motion is then compensated, that is, the sequences of frames are 'warped' to the GMOB [20]. Many VOS algorithms rely on camera motion compensated input. So, accounting for camera induced image motion in video sequences containing camera movement is a key step in MO segmentation.

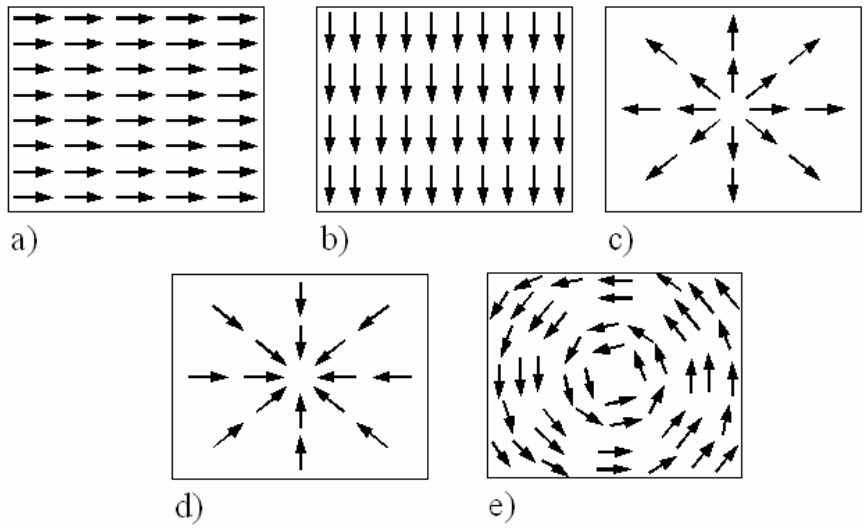


Figure 3.3 Motion vector patterns resulting from various camera operations: a) left pan; b) up-tilt; c) zoom in; d) zoom out; e) rotation

Modeling camera motion is concerned with describing the projection of movement on a 2-D plane [3]. Usually the 2-D projection does not describe the 3-D motion kinematics perfectly, since the 2-D motion projection lacks the depth information. However, when the scene viewed is planar or distant from the camera, the 2-D projection can be reliable [44]. In quantifying the camera movement, the 3-D camera operations which are discussed above need to be considered. The parameters or combination of them, which are estimated from the model are used to compensate the camera motion.

3.2.1 Motion models

Global motion can be estimated using different parametric models such as translation, transformation, affine or projective models. These models vary in computation complexity and ability [3]. The greater the complexity of the parametric model, the greater it's ability to model global motion.

The different parametric models are discussed as follows. The simplest global motion model consists of translation and isotropic scaling with 3 parameters. The translation can be modeled using two parameters and may only model a global shift (horizontal and/or vertical). The equation that describes scaling and translation is given as

$$\boldsymbol{V} = ax + \boldsymbol{b} \tag{3.2}$$

where $\boldsymbol{V}$ is the motion vector, a is a constant scaling parameter and $\boldsymbol{b}$ is the translation vector, $\boldsymbol{b} = [bx \quad by]^T$. Since the transformation model has only few parameters, it is not used to model complex combination of camera movements.

3.2.2 The affine motion model

The affine motion model is more general and can model anisotropic zooming, rotation and pure shear along with simple translation. Six parameters are required: 2 translation

parameters (for x and y directions) and a 2x2 matrix which describes the more complex models of movement. It is the most commonly used model for a number of advantages [3]. First, its linear structure and it is shown to provide similar results as the more complicated non-linear models. Second, it is simple and contains small number of parameters as compared to complex models. Also, is more resilient to noisy and sparse motion vectors [32]. However, affine model is robust under some assumptions: the scene is assumed planar (no perspective distortion), or, objects should be far away from camera (or background and stationary objects should cover more than 50% of the image area [47]).

The affine motion model describes the displacement of a pixel of a region from frame *n* to frame *n + 1* by translation, rotation and linear scaling. It is given by the following equations:

$$\begin{aligned} x' &= a_1 x + a_2 y + a_3 \\ y' &= a_4 x + a_5 y + a_6 \end{aligned} \tag{3.3}$$

where pixel (x, y) in frame *n* corresponds to (x', y') in frame *n – 1*. The parameters $a_1,\ldots, a_6$ are the global motion parameters which describe the model. These parameters can be estimated from motion vectors [3].

3.2.4 Projective motion model

The projective model is the most flexible model and requires eight parameters. It can be used to describe any form of 3D translation and rotation. This model requires 8 parameters and is given by

$$x' = \frac{a_0 x + a_1 y + a_2}{a_6 x + a_7 y + 1}, \qquad y' = \frac{a_3 x + a_4 y + a_5}{a_6 x + a_7 y + 1} \tag{3.4}$$

where $(a_0,\ldots,a_7)$ are the motion parameters, (x, y) denotes the spatial coordinates of a pixel (or of a block) in the current frame and (x', y') denotes coordinates of corresponding pixel in the reference/previous frame. The projective model is not linear and therefore finding parameters is difficult as compared to linear models.

3.3 Motion estimation (ME)

The methods for ME can be classified into two main groups [47]: 1) block-based matching [21],[23],[46],[14],[2] and 2) recursive methods [36],[45]. Recursive methods estimate motion between successive frames on a pixel by pixel basis, whereas block matching algorithms estimate motion on a block by block basis. Both estimation methods rely on the optical flow assumption (described in section 3.1). ME is one of the most computationally intensive operations in video segmentation. Hence, a number of researches have been done in this field [8],[2],[18],[23]. Block matching methods are the most widely used motion estimation methods for their low computational complexity compared with pixel recursive methods. Also, block matching methods are adopted by many video coding standards such as MPEG-2 or MPEG-1 and H.261.262/263 [9].

3.3.1 Block matching algorithms for motion vector estimation

Block matching algorithms (BMA) are the most widely used ME techniques for dense motion field estimation and in video coding (such as in standards ITU-T H.261/H.263 and ISO MPEG 1/2) because of their reduced computational complexity. In BMA, the current frame is divided into blocks of size $N \times N$ (usually 8x8 or 16x16) and due to the small size of blocks, the pixels of any block are assumed to undergo the same motion vector. The motion vector is found by searching for the best match in the reference/previous frame [47]. To find the best match, block matching uses a criteria which is minimizing a measure of matching error between the current block and blocks in previous frame. Typical criteria could be the least squares error or mean absolute difference. The mean absolute difference (MAD), which is given in equation 3.5, is the most popular matching criterion because it requires no multiplication.

$$MAD(m,n) = \frac{1}{N \times M} \sum_{i,j \in B} | I_k(i,j) - I_{k-1}(i+m, j+n) | \tag{3.5}$$

where the best estimate of the motion vector, (u,v) is defined to be the motion vector which minimizes $MAD(m,n)$ that is,

$$(u,v) = \arg\min(MAD(m,n)) \tag{3.6}$$

where I_k is the gray value of a pixel in the current frame and I_{k-1} is gray value of a pixel in previous frame

The block size can affect the motion vector estimation. Larger block sizes can contain more than one motion direction, while smaller size blocks may lead to incorrect matching. As the best compromise between the two, the most commonly used block size is a square block of 16x16 pixels [2],[31]. In some researches, variable block sizes have also been used [21].

A number of search algorithms have been proposed to calculate and compare the MADs of candidate blocks [31],[23],[51]. The simplest and the most straightforward method is the full search, which compares and calculates the MADs for all search positions. However, this straightforward method takes extremely large amount of computation although it can find the motion vector which gives the global minimum of matching error. As ME is a computationally intensive operation, researchers started looking for more efficient algorithms than the full search. However, there is a trade-off between efficiency and best global minimum of matching error. Keeping this trade-off in mind, a lot of sub-optimal algorithms have been developed. A review of some of these search algorithms is given below.

The main principle of (fast) BMAs is reduction of computation complexity. This is achieved by decreasing the number of search locations based on the assumption that the MAD is monotonically decreasing around the location of the optimal motion vector. Usually, the best match for a block is constrained within a search area up to 'p' pixels on all four sides of the corresponding block taken into consideration [2]. However, the MAD surface may include local minima, where the algorithms may be trapped. This will create final motion vectors different than optimal ones and a larger residual error in the motion prediction. Larger motions require larger search area and the larger the search parameter, the more computationally expensive the process of ME becomes [8]. Fast BMAs are used for ME in many video segmentation and coding schemes.

Full search: The full search algorithm exhaustively searches all the candidate blocks within a search area to find the best matching block. It gives the best results, but it is the most

computationally expensive algorithm. For example, for a search area of p=7 pixels (i.e. 15x15 search window), the algorithm has to make 225 comparisons to find the best matching block.

Three-Step Search (TSS): This algorithm starts with the search location at the center and searches at eight locations around the center block at a step size of 4, and also at the center block as shown in figure 3.4. From the nine locations searched so far it takes the best match that gives the least cost and makes it the new search origin. It then reduces the step size by half and repeats similar search for the remaining two steps until the distance/step size becomes one. The block found at that location with the least cost (smallest MAD) is taken as the best match. The number of comparisons needed here is 25 for a 15x15 search window, which is very much less than the full-step search [2].

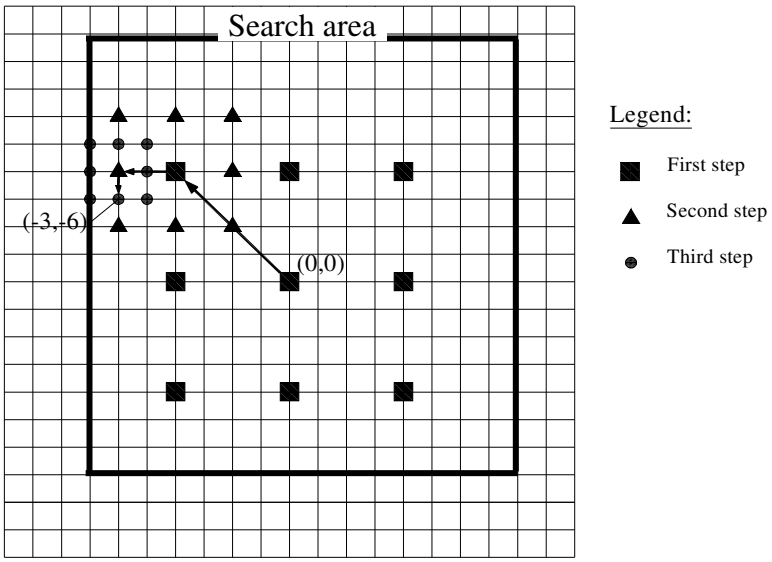


Figure 3.4 Three-step search. Motion vector for the best matching block is (-3,-6).

Logarithmic search: The Logarithmic Search (LS) is closely related to the three-step search (TSS). In each step, the MAD is evaluated on a grid of five pixels. From step to step, the distance between the search pixels is reduced in a logarithmic way. Using this technique, LS require more steps, searching in more candidate locations and consuming more time. However, it may be more accurate than the TSS, especially when the search window is quite large [53].

The LS algorithm can be described as in the following steps (figure 3.5) [51]:

1. Pick an initial step size. Look at the block at the center of the search, and the four blocks at a distance of 's' from this on the X and Y axes.
2. If the position of best match is at the center, halve the step size. If however, one of the other four points is the best match, then it becomes the center and step 1 is repeated.
3. When the step size becomes 1, all the nine blocks around the center are chosen for the search and the best among them (i.e. the block with the least MAD) is picked as the required block.

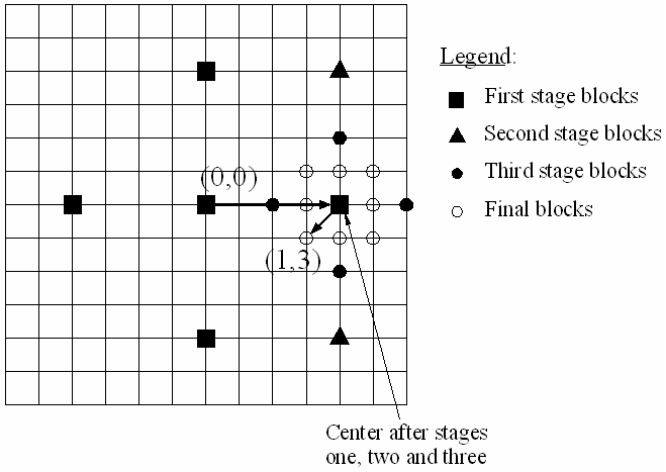


Figure 3.5 Logarithmic search. Motion vector for the best matching block is (1,3).

Hierarchical Search using mean pyramid images: Coarse to fine hierarchical searching schemes have been proposed to reduce computational complexity of search algorithms [31]. In the mean pyramid method, different pyramidal images are constructed by using low-pass filter to remove the effects of noise at higher level. Then hierarchical search motion vector estimation is performed, proceeding from the higher level to the lower ones. This reduces computational complexity and gets high quality motion vectors. Three level pyramidal images as shown in Figure 3.6 are constructed using simple averaging:

$$g_L(p,q) = \left\lceil \frac{1}{4} \sum_{u=0}^{1} \sum_{v=0}^{1} g_{L-1}(2p+u, 2q+v) \right\rceil \qquad (3.7)$$

$$1 \le L \le 2$$

where $g_L(p,q)$ represents the gray level at the position *(p,q)* of the *L*th level and $g_0(p,q)$ denotes the original image (at level 0), and $\lceil\ \rceil$ represents truncation.

A mean pyramid is constructed by simple non-overlapping low-pass filtering, by assigning the truncated mean gray level of pixels in a 2x2 low-pass window to a single pixel at the next level. Hence, one pixel at level-1 corresponds to a 2x2 block at level-0, and one pixel at level-2 corresponds to 2x2 block at level-1 and a 4x4 block at level-0. Therefore, a block size of 16x16 at level-0 is replaced by a one of size $16/2^L$ x $16/2^L$ at level-L.

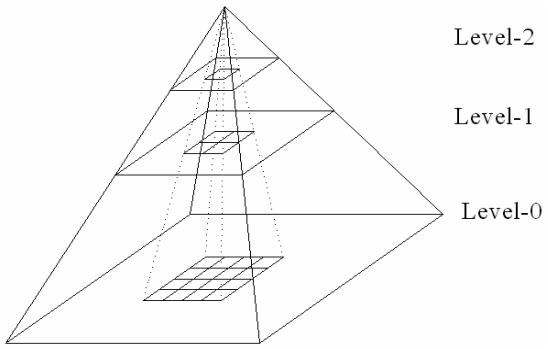


Figure 3.6 Mean pyramid construction

After construction of mean pyramid, the resulting images are used to search motion vectors starting from level-2 using MAD criteria discussed above. For a $I \times J$ sub-block, at level-*L*, equation 3.5 can be written as

$$MAD(m,n) = \frac{1}{IJ}\sum_{i=1}^{I}\sum_{j=1}^{J} | g_k(i,j) - g_{k-1}(i+m, j+n) \qquad m,n = 0,\pm 1 \tag{3.8}$$

where arguments *m* and *n* denote the relative displacement in the search area, and $g_k(i,j)$ represents the gray level at the position *(i , j)* in a sub-block on the *k*th frame. At level-*L*, *I* and *J* are replaced by $I'=I/2^L$ and $J'=J/2^L$ (L = 0, 1, 2) respectively, and eq.3.8 can be written as

$$MAD_L(m,n) = \frac{1}{I'J'}\sum_{i=1}^{I'}\sum_{j=1}^{J'} | g_{L,k}(i,j) - g_{L,k-1}(i+m, j+n) | \qquad L = 0,1,2 \qquad m,n = 0,\pm 1 \tag{3.9}$$

where L and k denote the Lth level and the kth frame respectively. The motion vector *(m,n)* with smallest $MAD_L(m,n)$ is selected as the coarse motion vector at level-L, which is then propagated to the next lower level and used as initial vector for refined motion search at that level. The process is repeated down to level-0. If the motion vector at level-L is represented by $MV_L(r,s)$, the detected motion vector at level-L cam be written as

$$MV_L = 2 \times ML_{L+1}(r,s) + \Delta MV_L(r,s), \qquad L = 1, 0 \tag{3.10}$$

where (r,s) is the position of the search block and $\Delta MVL(r,s)$ is the updated increment of motion displacement at level-L. In this case, referring to Figure 3.6, one-pixel interval at level-2 corresponds to four-pixel intervals at level-0, and one pixel interval at level-1 corresponds to two-pixel intervals at level-0. The final motion vectors are then the motion vectors MV_0 obtained at level-0, which is the original image.

Selection of block size is an important criterion, as explained above, that determines the quality of the resulting motion vectors. Generally, larger blocks are suitable for rough but robust estimation [31] while smaller blocks are suitable for localizing the estimation. However, they are susceptive to noise. A good compromise is a 16 x 16 pixel block at level-0, which is the most commonly used block size. Since MADs are computed at the highest level based on relatively small blocks, almost the same values are likely to appear at several points. Thus to solve this problem, [31] suggested to use more than one candidate at the highest level. A number of motion vectors at the highest level are propagated to the lower one. Three-step search or full search with one or two pixel resolution in a small window around the candidates can be used at each level to find the minimum MAD. Figure 3.7 shows search locations of the algorithm.

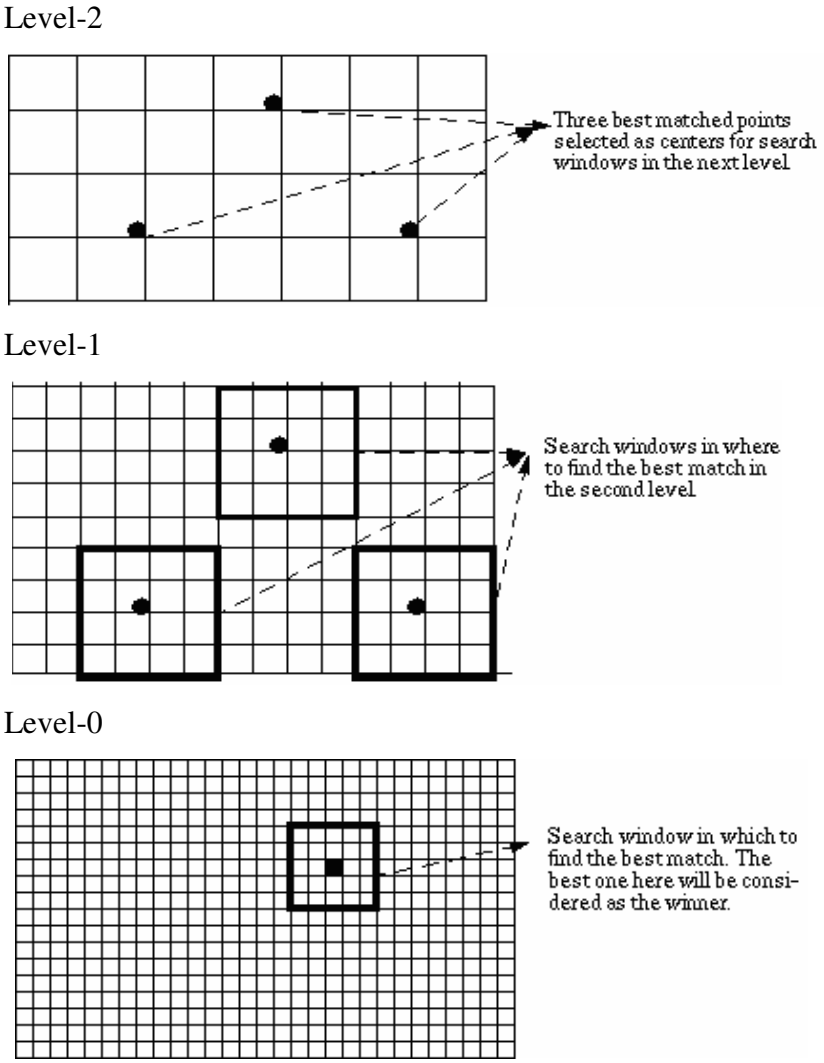


Fig. 3.7 Search locations for the hierarchical search algorithm [51]

3.3.2 Parameter estimation

In the parametric global ME, the parameters of the particular motion model used can be estimated from motion vectors. Estimation of parameters involves minimizing the error between the motion vectors of the dense and parametric model. The most commonly used error minimization is the least squares method. For the affine model used for example, the error to be minimized can be given by:

$$E = \sum (V_x' - a_1 x - a_2 y - a_3)^2 + (V_y' - a_4 x - a_5 y - a_6)^2 \tag{3.11}$$

There are many different methods for parameter estimations. Some approaches support direct estimations of model parameters [45]. The most commonly used methods, first estimate the dense motion field and then by regression or by the least squares method, fit the model to the dense motion fields [21],[11],[47]. A review of some of the approaches is given below.

To recover global motion parameters from motion vectors, [46] derives a motion model first. The model contains both local motion due to object movement and the global motion due to camera zoom and pan (translational and rotational pans). The model is taken from the image projection geometry that describes the effect of object and camera motion in recorded video. By examining the motion vectors, object and global motions are identified. Then the global motion parameters, the zoom factor and combined pan factor are estimated from the expression of image projection geometry.

The authors in [21] proposed adaptive robust estimation of affine parameters from block motion vectors. The robust estimation approach, which uses M-estimators, is based on outlier rejection scheme. In this method, a merit function is set that measures the agreement between the velocity vectors and the model with a particular choice of parameters. The merit function is conventionally arranged so that small values represent close arrangement. The parameters of the model are then adjusted to achieve the minimum in the merit function. The adjustment process is a problem of minimizing the residual error with respect to model parameters by filtering out suspected outliers of input motion vectors. A continuous weight function based on a sigmoid function is used to filter out potential outliers. In terms of accuracy, this approach produces better results. However, tuning weight functions adds computation intensity to the estimation process.

Robust estimation of affine model parameters: In order for global motion model to best approximate camera movement, it is necessary to identify areas of the picture where motion vectors describe global motion. There are also areas where motion vectors describe object motion or which result from errors in motion vector estimation. These areas, which are called outliers has to be removed by outlier suppression technique. Different rejection criteria can be used: removal of motion vectors at picture edges, motion vectors with large MAD values, motion vectors which differ greatly from their neighbors, or removal of large motion vectors.

3.4 Camera motion compensation

Once optimal affine parameters are estimated, the next step is compensating the camera motion according to the motion model considered. The process of compensating the camera motion of a frame with respect to a GMOB is called *frame warping*. Frame warping consists of aligning a frame with respect to a GMOB. New coordinates of each pixel of the frame to be warped are computed using the equations of the motion model. The pixel positions computed may not be integer values. For these non-integer values, pixel intensity values are computed using interpolation.

Different methods of interpolation are available such as nearest neighbor, linear, bilinear, bicubic, spline, quadratic, etc. Nearest neighbor is the most basic and requires the least processing time of all the interpolation algorithms. This is because it only considers one pixel – the closest one to the interpolated point. The linear interpolation takes the mean average of two adjacent points to find the mid point. The bilinear interpolation, which is the most commonly used method, extends the linear interpolation for two dimensions. Linear interpolation is performed in one direction first, and then in the other direction. It considers the closest 2x2 neighborhood of known pixel values surrounding the unknown pixels. It then takes a weighted average of these 4 pixels to arrive at its final interpolated value. This results in much smoother looking images than nearest neighbor. If the values of a function f at four points $Q_{11} = (x_1, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$, and $Q_{22} = (x_2, y_{2)}$ are known, bilinear interpolation can be estimated as

$$\begin{aligned} f(x, y) \approx & \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y) \\ & + \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1) \end{aligned} \tag{3.12}$$

Literature has shown that a simple bilinear interpolation achieves good performance for half-pixel accuracy in traditional block matching motion compensation methods. However, to further increase accuracy beyond half-pixel in these systems, more sophisticated interpolation methods like bicubic and spline can be used, if computation intensity of these algorithms could be tolerated.

To summarize, this chapter discussed and compared the techniques used in global motion estimation and compensation, which is a serious problem in using CD based AS systems. The next chapter discusses the CD itself, which is the main part of the system which is proposed for AS.

Chapter 4
CHANGE DETECTION

4.1 Introduction

Generally, change detection is a process of identifying the difference in the state of objects or phenomena by observing them at different times [5]. For example, it can detect moving cars in road traffic surveillance systems. Change detection is the main part of the automatic video segmentation techniques used in this thesis. Thus, studying efficient and accurate technique that can detect and label changes automatically in the video sequence between different time instants is the main concern of this chapter.

Change detection is used in a number of applications including video surveillance, remote sensing, medical diagnosis, driver assistance and video segmentation. However, the application is limited to video segmentation in this discussion. The goal here is to identify the set of pixels that are significantly different between an image and another reference image in video sequences. A comparison is applied between the image under concern and the reference image to detect those pixels. The changes may result from (1) camera movement, (2) appearance/disappearance of objects, (3) motion of object relative to background, (4) shape changes of objects, or (5) changes in illumination. These changes manifest as temporal intensity variations at each corresponding pixel between the two images. However, not all the resulting changes correspond to the objects to be segmented. The change detection algorithm should distinguish the temporal variations caused by MOs from noise or other causes. The set of pixels that are labeled as change constitute a change mask, which is a binary image that represents the objects of concern. A decision on when to label a change is a crucial point in change detection. The decision is made by comparing the changes against a threshold (see section 4.4 for thresholds) which is determined empirically or automatically. A change mask should not contain unimportant forms of change such as those induced by camera motion,

sensor noise, uncovered backgrounds and illumination variation. [5] states that accuracy in the detection of object contours (spatial accuracy) and temporal stability of the detection (temporal coherence) are the general requirements for change detection algorithms. Moreover, sensitivity and robustness are desirable features. Sensitivity refers to ability of detecting changes of small magnitude. Robustness can be seen as the property of providing good accuracy under varying conditions, for example illumination changes. There are a number of researches (as discussed in next sections) that consider the sensor noise and illumination variations to produce near to accurate change masks. However most of the approaches take the assumption that camera is fixed, or accept camera motion compensated frames as input (i.e. fixed GMOBs is assumed).

As far as the camera motion is concerned, two frameworks can be identified. If the camera is static, change detection aims at recognizing MOs (foreground) and the static background. If the camera moves, the goal of change detection is to recognize coherent and non-coherent moving areas. Coherent moving areas correspond to background, and the non-coherent to MOs [5]. For videos with moving camera, reference image is usually generated by estimating camera motion. The reference image is then aligned with the current image before change detection is performed.

Presence of motion usually causes three kinds of “change regions” to appear: 1) uncovered background 2) covered background 3) overlap of two successive object projections. The uncovered background (discussed in section 2.2 of chapter 2) is the portion of background which will be visible when an object in the scene moves. Temporal change occurs in this area also, which needs to be avoided. The covered background and overlap of two successive object projections constitute the object to be detected in the frame of concern. In cases where object surface intensity is uniform, overlapped part is difficult to recover by temporal change detection. Only areas near edges of the surfaces are detected and the center of surfaces forms holes in the objects detected. This is one drawback in change detection based systems [43] and it requires intensive post-processing. Usually, the results of change detection based segmentation algorithms are rough, and object boundaries may be irregular. The detection of

some noise will also produce small holes inside the object and small regions outside the object. These problems can be easily treated by applying post-processing.

The next section presents a review of the different change detection methods in literature. In section 4.3, general model for change detection based on [5] is explained.

4.2 Segmentation approaches based on change detection

A number of researches have been done in the area of segmentation based on change detection to identify the presence of motion in video sequences considering both moving and fixed backgrounds [1],[6],[28],[7][10],[52]. These researches also aim at solving the problems in change detection based systems. In some approaches (e.g. [52]), areas that do not follow the coherent (background) motion are considered as indicators of presence of independently moving physical objects. The change areas are obtained during motion field estimation for global motion compensation (as outliers, for example, as explained in section 3.3.2). However these changed areas do not provide good initial semantic partition of MOs, since only parts of MOs are detected.

A popular approach to change detection based segmentation usually compares the current frame with a GMOB to detect changes in the current frame. The GMOB could be previous or background frame. In a background subtraction algorithm, the current input image is compared with the background image and foreground objects are detected at places where the differences between both images are large. In some approaches, it is assumed that a picture of the scene background is available which does not show any foreground objects. It is also assumed that background is usually stationary or has a simple global motion and that can be updated for any variations that may be due to illumination or noise. The simplest method to compare the current frame with a GMOB to detect changes is frame differencing and applying thresholds. Other methods such as higher order statistics and image ratioing can also be used.

In a paper by Chien et al. [10] change detection is used as the basic idea for MO segmentation from the background. The authors aim to solve the problem of uncovered background and to reduce shadow effects. A method called background registration technique is used to construct a background image from accumulated frame difference information. The MO region is then separated from the background region by comparing the current frame with the constructed background image. However, this approach assumes stationary background video. The background registration technique which combines several past frames to produce reliable background also requires memory element and more processing time for maintaining registration masks.

Chen et al. [7] proposed video object segmentation based on change detection and background updating. It combines frame difference mask and background subtraction mask to acquire initial object mask and solve uncovered background problem. The change detection is used to analyze temporal information between successive frames. It separates difference frame into changed and unchanged regions according to a threshold obtained from background estimation. However, this algorithm does not present how the initial background is obtained. It also assumes global motion compensated input. [28] uses background updating to achieve robustness to illumination changes in the scene for the detection of changes in outdoor video surveillance images.

The performance of change detection algorithms is highly dependent on selection of thresholds discussed in section 4.4. Various thresholding techniques can be applied from empirically determined to adaptive ones. Simple approaches to motion detection consider thresholding techniques on pixel by pixel, or block wise difference frames to improve robustness against noise. More sophisticated models have been considered within a statistical framework, where the inter-frame difference is modeled as a mixture of Gaussian or Laplacian distributions.

The proposed method in [28] uses a pixel-based difference technique for the change detection module. The system decides that a pixel is changed if the difference between the gray-level

value of that pixel in the current image, *D(x,y)* and the correspondent pixel in the background image is higher than a certain threshold T. This decision is given by the following rule:

$$D(x, y) = \begin{cases} 0 & if \mid I_n(x, y) - I_{n-1}(x, y) \mid < T \\ 1 & if \mid I_n(x, y) - I_{n-1}(x, y) \mid \geq T \end{cases} \quad (4.1)$$

The value of the threshold T can be determined empirically through a series of experiments (which is the most common approach). However, different values have to be determined as the type of input sequence changes. This approach is therefore not suitable for automatic segmentation. For automatic and optimal detection of changes, the threshold has to adapt to the scene content and to the different kinds of noise [6].

Although segmentation based on change detection has a number of drawbacks, it is more efficient than other types, and also less computation intensive. Therefore it is mainly preferred in automatic segmentation. Also, most of the problems in change detection are simple, and can easily be dealt with.

4.3 System model for change detection

A basic change detection algorithm takes the image sequence as input and generates a binary map $C(x, y, n)$. This binary map represents the pixels at *(x,y)* of an image $I(x, y, n)$, which is the current frame *n* under test with respect to a another image $I(x, y, r)$ which is the reference frame *r*, and is defined as:

$$C(x, y, n) = \begin{cases} 1 & if\ change\ occured\ at\ time\ n; \\ 0 & otherwise \end{cases} \quad (4.2)$$

A general structure for change detection has been proposed in [5] for computing $C(x, y, n)$ as a function of $I(x, y, n)$ and $I(x, y, r)$. This general structure is shown in the following block diagram (Fig.4.1) and discussed in the subsequent sub-sections.

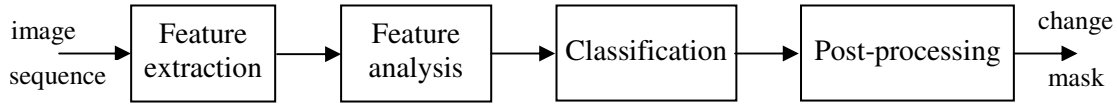


Figure 4.1 Block diagram of general structure for change detection [6]

4.3.1 Feature Extraction

In the feature extraction stage, the input frame sequence is transformed into a most appropriate feature space. This feature space may represent luminance, color components, or more complex feature spaces. Luminance is the most commonly used feature space. In color video, luminance can be computed as weighted combination of color components. For the RGB color space for example, the luminance *I(x,y)* can be computed as

$$I(x, y) = w_1 R(x, y) + w_2 G(x, y) + w_3 B(x, y) \tag{4.3}$$

Where w_i's are weights which account for sensitivities of the human visual system to color components. Color information can also be used directly, or by converting to other color spaces. Some color spaces (e.g. YUV, CIE L*a*b*) provide advantage of illumination invariant features so that fluctuations of intensity values will not produce false positives. This adds robustness to change detection under varying illumination.

4.3.2 Feature Analysis

In feature analysis, the current image and reference image derived after feature extraction are compared to detect areas of change. Different ways of comparing can be implemented.

The most commonly used is pixel-wise differencing, due to its simplicity [39], given by

$$d(x, y) = I_n(x, y) - I_r(x, y) \tag{4.4}$$

where $I_n(x, y)$ is the current image and $I_r(x, y)$ is the reference image. Image ratioing, vector difference, or higher-order statistics are other ways of comparison. To reduce the effect of noise, the comparison is made inside a small rectangular window which slides over the images. The window size is selected so as to have trade-off between robustness to noise and to detection accuracy.

Preprocessing may be needed to be applied to filter out common types of unimportant changes before change decision is made. This involves geometric and intensity adjustments [35] as explained below.

Geometric adjustments: include filtering apparent intensity changes at a pixel resulting from camera motion, and image registration (alignment of several images into the same coordinate frame) is applied for filtering.

Intensity adjustments: for changes in strength of light sources in the scene. These adjustments can be made by *intensity normalization* – for illumination-invariant change detection. Pixel intensity values are normalized to have the same mean and variance as those in another i.e.

$$\bar{I}_2(x, y) = \frac{\sigma_1}{\sigma_2}\{I_2(x, y) - \mu_2\} + \mu_1 \tag{4.5}$$

where $\bar{I}_2$ is the normalized second image and μ_i and σ_i are mean and standard deviation of intensity values of I_i respectively. Alternatively both images can be normalized to have zero mean and unit variance. This allows the use of decision thresholds that are independent of original intensity values of images. Images can be divided into corresponding disjoint blocks, and normalization independently performed using local statistics of each block to achieve better performance at the expense of introducing blocking artifacts.

Perhaps the most critical issue in comparison is selection of the GMOB. Two choices are possible: the previous frame relative to the current frame in the sequence, or an image representing background of the scene. The background frame can be fixed, or can be updated periodically. Fixed background frames can be obtained from the sequence prior to entrance of

MOs to the scene. Background can be updated periodically [28] by temporal integration of previous and next frames, to produce large background image as in [19]. However, those approaches are not always practical for the following reasons: For using fixed backgrounds, there should be an initial frame of a sequence with no foreground objects which is not always available. Construction of large background image complicates the system and introduces delay, since it requires integration of a number of previous and next images to obtain optimal background image.

The above problems of using background frame as reference can be solved by background updating (as discussed in approaches in section 4.2) or using the previous frame as reference. Many change detection techniques use the previous frame as reference [25], [10], [43]. This offers the advantage of reducing detection of shadows. On the other hand, this approach introduces the uncovered background problem [5]. Further more, it does not detect low textured moving areas of objects. However, addressing these problems may be easier as compared to background updating.

4.3.3 Classification

To obtain the binary map, based on the results of comparison in the above step, a pixel under test is classified into one of two classes: changed or unchanged based on equation 4.2. However, since the results of comparison step contain noise, the decision is done by comparing the results against a threshold. This threshold may be set empirically or computed adaptively. Whenever this threshold is exceed, the pixel marked as changed. Thus, equation 4.2 becomes, for example, for a difference image $d(x, y, n)$:

$$C(x, y, n) = \begin{cases} 1 & if\ d(x, y, n) > T; \\ 0 & otherwise \end{cases} \tag{4.6}$$

where T is the threshold value. Determination of optimal decision thresholds is crucial. To obtain robust classification, error probability should be minimal [1]. Too low threshold values will detect spurious changes, while high values suppress significant changes. Proper value of threshold is dependent up on the scene, camera noise and conditions that may vary

over time (e.g. illumination). Early change detection methods were based on simple differencing and applying simple fixed thresholds to the difference image. Often, the threshold is chosen empirically. However, empirically determined thresholds have the limitation that the threshold values have to be interactively tuned according to the scene characteristics. This is not suitable for automatic applications since users should interact to set the threshold values. Automatically determined thresholds that adapt to the scene content and the different kinds of noise are required for these applications. Different approaches to adaptive threshold computation are given in the following section. The decision rule in many change detection algorithms is cast as a statistical hypothesis test. This is done to obtain adaptive and automatic thresholds.

4.4 Adaptive threshold computation

Thresholds can be linked with time-varying phenomena (e.g. illumination, noise) which interfere with detection of changes so that they can be obtained dynamically. The most common approach to performing the automatic thresholding is to assume particular distribution models for the difference of image samples and the noise [40]. If the probability density function (pdf) of the phenomena is known, an adaptive threshold can be computed by applying statistical hypothesis tests [5], [1]. For example, the probability density function of the camera noise is assumed to be Gaussian, since it is white.

Rosin [39] considers two modeling approaches for change detection: either modeling signal and/or noise, or modeling intensity and/or spatial properties. Differencing is applied to edge maps rather than intensity images. He then modeled the noise in the edge maps using Rayleigh distribution given by

$$R(x) = \frac{x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}}; \quad x \geq 0 \tag{4.7}$$

The Rayleigh function is then approximated by Normal distribution and thresholding is applied at xs. This threshold is chosen for a given acceptable proportion of false motion

pixels, which is determined from the probability of incorrectly classifying a pixel as motion given by

$$P_F = erfc\left(\frac{x}{\sqrt{2}\sigma}\right) \tag{4.8}$$

where *erfc*(.) is the complementary error function. To estimate the noise variance s^2, Rosin used Least Median of Squares (LMS) applied to the difference image histogram. According to Rosin, the LMS and the expected standard deviation of the noise are related by

$$\sigma = \frac{LMS}{0.33724} \tag{4.9}$$

Rosin also described modeling the signal intensity using a different distribution. Since little is known about magnitudes of the difference image intensities produced by change, original images are analyzed instead. Windows surrounding two corresponding pixels from the images considered are compared using Kolmogorov-Smirnov test. Thresholding is performed by accepting as motion only those pixels whose distributions are dissimilar, i.e. their test statistic is above the critical value for a selected significance value (5%). Spatial properties of the noise and signal also are modeled by different distributions. Rosin found out that modeling based on intensity distributions did not work well. Although spatial methods were promising, their robustness was not determined and is difficult to apply them.

Aach et al. [1] proposed a method of determining decision threshold which is related to false alarm rate, by using hypothesis testing, in particular, significant testsd. The hypothesis is that some observed temporal intensity variation is caused by noise only (the null hypothesis – H_0), and that statistical properties of this hypothesis are assumed to be known. The objective of this hypothesis is that a site on the image grid shall be marked as changed when evidence collected from temporal gray level difference does not support the assumption that observed variation is not arising from noise. instead of applying the decision of changed or unchanged on a single pixel of the difference image, a window of pixels is considered to make the decision more reliable Three different statistics were investigated inside a small window: 1) local sum of squared normalized differences assumed white Gaussian camera noise (the pdf given by equation 4.9), 2) local sum of absolute gray level difference with assumption of

Laplacian noise, and 3) local average of differences inside the window. The first statistic is discussed as follows.

Under the assumption of H_0, the intensity difference image d_k obeys a zero mean Gaussian distribution N(0,s) with variance s^2, that is,

$$p(d_k \mid H_0) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{d_k^{\,2}}{2\sigma^2}\right\} \tag{4.10}$$

Since the camera noise is uncorrelated between different frames, the variance s^2 is equal to twice the variance of the assumed Gausian camera noise distribution. Since $p(d_k|H_0)$ depends only on the squared ratio of the difference d_k normalized with its standard deviation, the decision on the label of pixel k can be based on this squared ratio. The decision can be made more reliable by using a small window surrounding the current pixel. The local sum of squares $\Delta^2=(d_k)^2/s^2$ is then calculated inside the window. Since (d_k/s) obey a zero-mean Gaussian distribution, the sum Δ^2 obeys a Chi-distribution which has as many degrees of freedom as there are pixels inside the window. With this distribution known, the decision between changed and unchanged can be arrived at by a significant test by specifying a significance level α and a corresponding threshold t_α is computed as

$$\alpha = \text{Prob}(\Delta^2 > t_\alpha \mid H_0) \tag{4.11}$$

Whenever Δ^2 exceeds t_α, the corresponding pixel is marked as changed, otherwise unchanged.

Gachter [15] also modeled the noise by a normal distribution N(0,s^2) with zero mean and described by the standard deviation s. s is estimated by the least median of squares of absolute difference images. Other statistical tests have also been used, like likelihood ratio tests and probabilistic mixture models [35].

4.4.1 Noise variance estimation

The noise variance is related to the variance of camera noise. It can be estimated offline for the camera system used, or directly from the difference image on those areas where the

hypothesis H_0 is valid. It can also be estimated from the entire frame difference image if those areas cannot be determined easily. In this case, since difference images contain not only noise but also foreground objects a robust estimation technique for the standard deviation is required. The least median of squares rests unaffected while the moving part in the image constitutes less than half of all image points [15]. It is computed by determining the centroid of the shortest range in the sorted sample $\{a_k\}_{m.n,k=1}$ $i,e\ a_1 \le ...a_k \le ... \le a_{m.n}$ includes all image points of the absolute difference image.

$$a_{\mathrm{LMS}} = \frac{1}{2}(a_{k_{\min}} + a_h + k_{\min}) \tag{4.14}$$

where $k_{\min} = \arg\min_k (a_{k+h} - a_k)$ *and* $h = \left[\frac{m.n}{2}\right] + 1.$

Differencing noisy images followed by taking absolute values produce the normal distribution $2N(0, 2\sigma^2)$ for positive values only. The least median of squares for this distribution is 0.3372 (according to [15]) and thus the standard deviation is determined by $\sigma = a_{\mathrm{LMS}} / 0.3372$. However, estimating noise from the entire frame difference image is computationally expensive, since it involves sorting the difference values.

4.5 Post-processing

As mentioned in the above sections, the result of change detection algorithms is not accurate. Due to noise and uncovered background, there exist some noise regions in the initial object mask in both background region and object region. Also object boundary is not very smooth. Therefore a post-processing step to eliminate these noise regions and filter out the ragged boundary is necessary. The post-processing can be applied either to the binary mask only or to both the binary mask and original frame [5].

Post-processing using binary mask only is the simplest approach to remove irregularities. It preserves contours and reduces spurious regions of the mask. It does not use any information from the original image. The post-processing may be applied to the current result or a set of

results. It consists of simple morphological opening or closing (Appendix – A), or a more complex composition of morphological filters.

Morphological opening and closing for post-processing: [4] (Appendix-A) Opening has the effect of eliminating small and thin objects breaking objects at thin points, and generally smoothing boundaries of larger objects without significantly changing their area. Closing has the effect of filling small and thin holes in objects, connecting nearby objects and generally smoothing boundaries of objects without significantly changing their area. The open operation is effective for removing noise within the object region and the close operation is effective for eliminating the background noise. However, noise regions whose areas are larger than the structuring element cannot be removed by the close or open operations. In order to remove noise regions with large area, larger structuring elements should be used. This will not only increase the computation complexity but also degrade the precision of the object boundary. Successive openings and closings can improve the situation markedly.

The use of binary masks in post-processing has the advantage of reduced computational cost. However, when the MO does not have sufficient textures, the result of change detection will be such that larger holes will be created inside the uniform regions of the object. Filling these regions requires repeated application of closing, which itself produces incorrect boundaries by adding unnecessary pixels at the boundaries. Also, if the holes produced are so large, any number of close or open operations may not fill them while preserving exact object shape. In such situations, it will be advisable to integrate the original frame with the binary mask. Some spatial information (e.g. neighborhood, spatial gradient) of the original frame can be used to identify the smoothness of object regions and this can be integrated with the binary mask to get better results.

In summary, this chapter discussed the main part of the automatic SVO segmentation process which is proposed in this thesis. The next chapter presents the design of the proposed system.

Chapter 5
PROPOSED SYSTEM DESIGN

5.1 Block diagram of system

Design of a fully AS system is proposed in this chapter. From the discussions presented in chapters 3 and 4, efficient and simplified techniques are selected for the main stages and are combined for the proposed effective automatic SVO segmentation system. The design considers videos with fixed and moving backgrounds to make the system more flexible. Depending on whether there exists camera motion or not in video sequences, and whether there is an initial background, three cases are examined in the design of the segmentation system: Case-1 is for video sequences where there is no camera motion, or when background frame is fixed, and also when there exists initial background frame that contains no MOs. Case-2 is for video sequences where there is no camera motion, but when there is no initial background that can be obtained. Case-3 is for video sequences where there is camera motion, and hence when there is GMOB. Block diagram of the entire proposed system is shown in Figure 5.1, and the description of overall working for each of the cases is given below. The description of each block and discussion about selection of the particular algorithms used in each block is presented in the next sections.

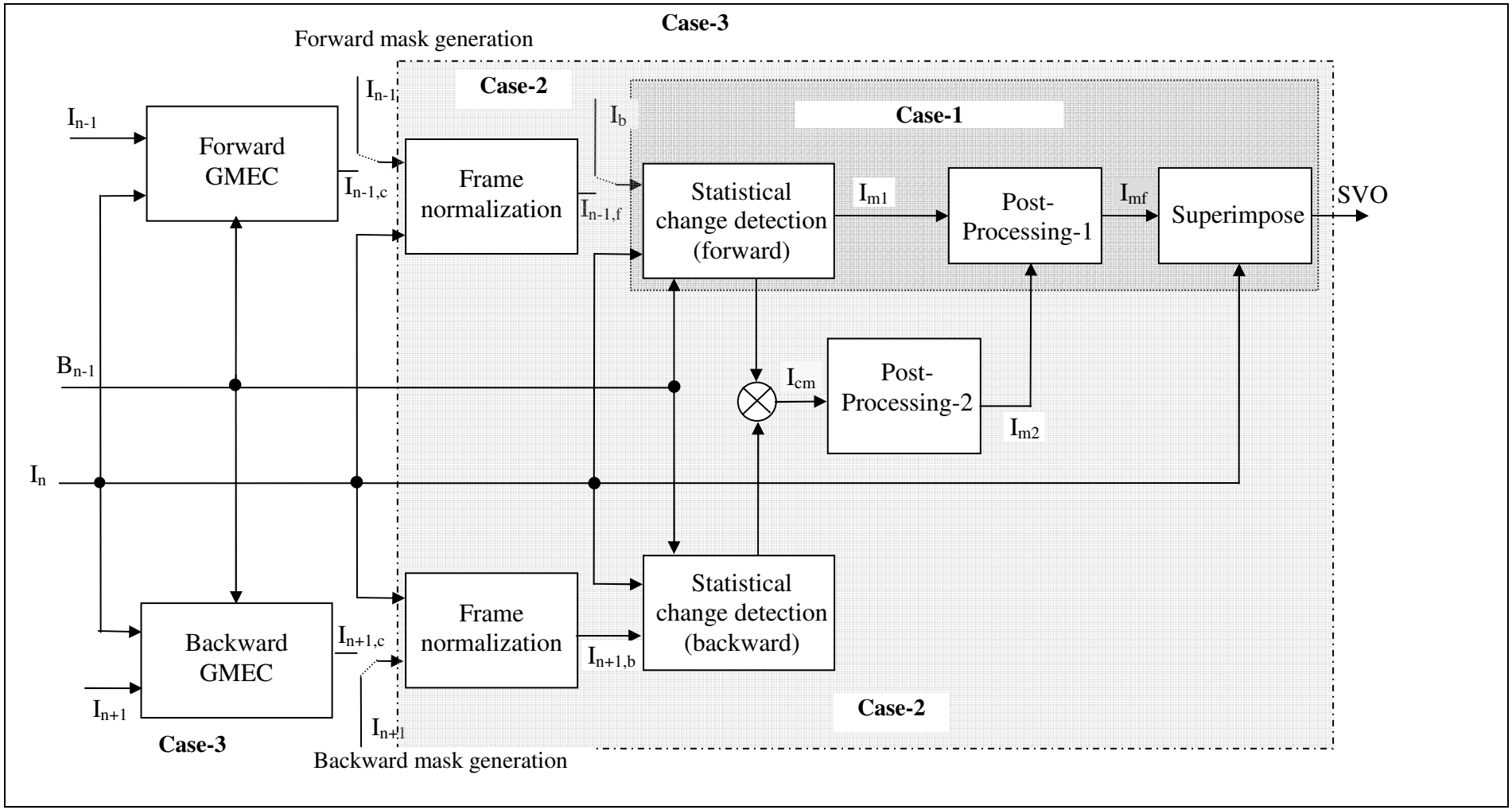


Figure 5.1 Block diagram of the proposed system.

Case-1: No GMOB, with initial background

This case considers applications where an initial background GMOB can be easily obtained. The system uses the background GMOB (I_b), the current frame (I_n), and the previous background frame B_{n-1} to obtain the SVO in the current frame (the dark-gray shaded region in Figure 5.1). First, an effective statistical change detection based on significance testing (presented in section 5.3) is applied (the **Statistical change detection** block) between I_n and I_b to obtain a binary SVO mask I_{m1}. The change detection algorithm uses the reference background frame (B_{n-1}) to estimate the background noise used in significance testing. Then, a post-processing stage – **Post-processing-1** (discussed in section 5.4) is applied to the output of the change detection block. The output of this stage is the final object mask, I_{mf}. The final binary object mask is then superimposed on the original frame in the next stage (**Superimpose** block) to produce the final segmented SVO and background frames. Note that the result of Frame normalization step, $I_{n-1,f}$ is not used in this case, instead, the system switches to using I_b.

Case-2: No GMOB, without initial background

In this case, the architecture of the system aims at removing uncovered background by integrating two change detection masks (obtained by forward and backward mask generation). These change detection masks are combined by a logical operator. The operator removes all areas except the foreground object detected which is the region that overlaps in the two change detection masks. This can be explained using Figure 5.2 as follows. Consider the two change detection masks (forward and backward) shown in Figure 5.2 a) and b). Region A is uncovered background revealed during forward mask generation, and region C is uncovered background revealed by backward mask generation. Region B is the object which is detected in both change detection masks. When these two masks are combined using a logical AND operator, the result will be the one as shown in Figure 5.2 c). As this figure shows, the uncovered background in both change detection masks is removed in the final mask, and the remaining region, region B in Figure 5.1 c) is the required object mask in the current frame.

In this case, the system uses three consecutive video frames, the previous frame I_{n-1}, the current frame I_n, and the next frame I_{n+1} and also segmented background mask of the previous frame (B_{n-1}) as inputs. Using these inputs, two change detection masks (forward and backward) are produced as shown in the light-gray shaded portion of Figure 5.1. In the forward mask generation process, first the previous frame I_{n-1} is normalized with respect to the current frame I_n. This **Normalization** step is applied for feature analysis (section 4.3.2) to reduce the effect of illumination variations. Then, the same change detection used in Case-1 is applied between the normalized previous frame $I_{n-1,f}$ and I_n. In the backward mask generation process, the frame I_{n+1} is normalized with respect to I_n and the same change detection is applied to the normalized $I_{n+1,b}$ and I_n frames. Next, the results of forward and backward mask generation are combined by using a logical AND to produce a combined mask I_{cm}. Two stages of post-processing are applied in this case, in the following order: the stage **Post-processing-2** is applied first to I_{cm}, the combined mask by integrating with the current frame. Then the stage **Post-Processing-1** is applied to the result of **Post-processing-2** to produce the final object mask I_{mf} (the arrow from **Statistical change detection** block to **Post-processing-1** block works for Case-1 only). This final object mask is then superimposed on the original frame in the next stage, which is the **Superimpose** block to produce the final segmented SVO.

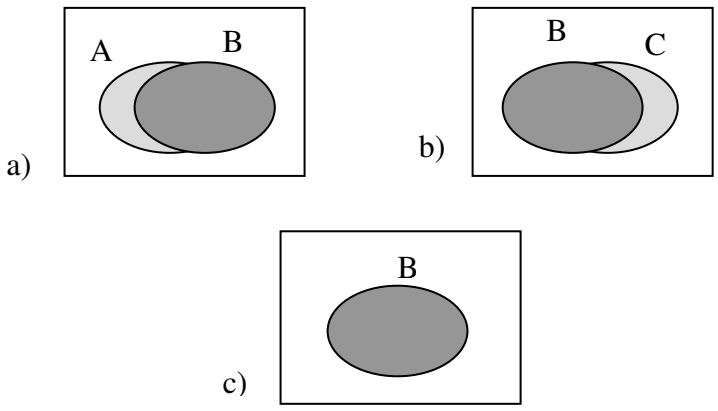


Figure 5.2 Removal of uncovered background by combining two change detection masks a) forward change detection mask; b) backward change detection mask; c) combination of a and b.

Case-3: With GMOB

For video sequences with moving backgrounds, the system first compensates the global motion. The rest of the segmentation process is just similar to Case-2. The forward and backward Global Motion Estimation and Compensation (GMEC) are applied to align previous and next frames, I_{n-1} and I_{n+1} respectively, towards the current frame I_n. These two blocks are the **Forward GMEC** and the **Backward GMEC** blocks in Figure 5.2. Block based ME, discussed in section 5.2 is used for the purpose of GMEC. It is assumed that in video sequences with moving backgrounds, obtaining an initial reference background frame is very difficult and so the system combines three consecutive frames just similar to that of Case-2.

5.2 Global motion estimation and compensation (GMEC)

For video sequences that contain camera motion, which causes GMOB, this global motion is compensated by the Motion estimation and compensation blocks. These are the **Forward GMEC** and **Backward GMEC** blocks in Figure 5.2, as explained in Case-3 of section 5.1. There are three phases of the motion estimation and compensation process proposed in this design:

1) dense motion vector estimation
2) parameter estimation
3) motion compensation or frame warping

Dense motion vector estimation: For the dense motion vector estimation, a three-level hierarchical block based algorithm described in section 3.3.1 of chapter 3 is used for its compromised performance in computational complexity, speed and accuracy. Mean pyramids are constructed using the simple averaging equation (equation 3.7). Then the pyramidal images are searched from top to bottom using MAD (equation 3.5) for block matching criterion. A full search within two pixels distance around the concerned block is done for better accuracy of matching block. The most commonly used 16x16 pixels block size is used at level-0 of the hierarchy. The process is shown in Figure 5.3.

Parameter estimation: The six parameter motion model, the affine model defined by equation 3.3, is used for the global motion. For estimation of the affine parameters, the least squares minimization criterion (equation 3.11) over the background frame is used to minimize the error between the dense and parametric model motion vectors. The previous background mask (B_{n-1} in Figure 5.2) is used to estimate the background part of the current frame where the global motion is valid.

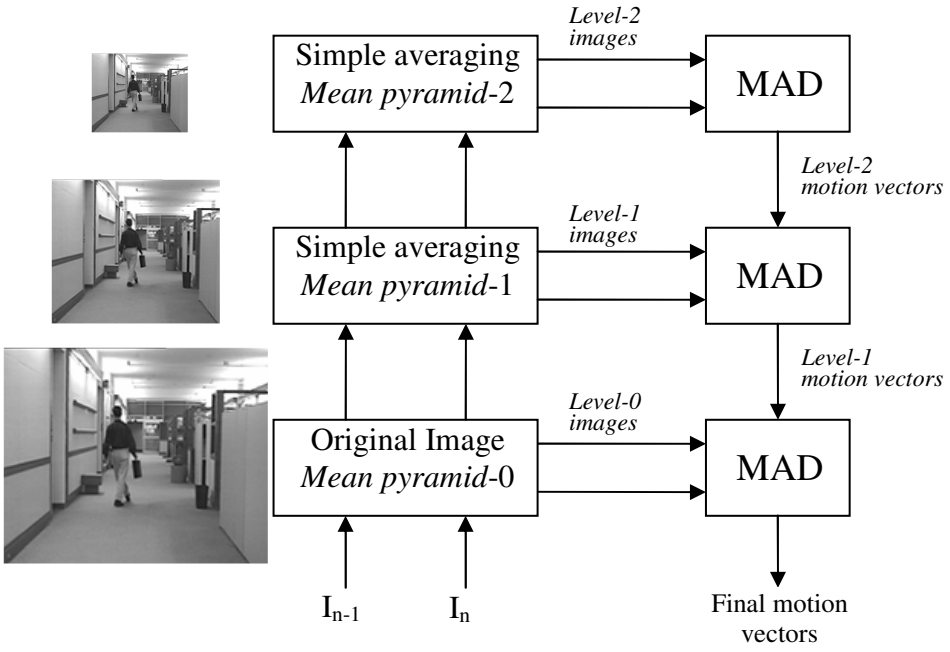


Figure 5.3 Three-level hierarchical mean pyramid based ME

Outlier suppression: Parameter estimation is sensitive process and for optimal parameter estimation, reliable motion vectors should be used in the estimation process. First, motion vectors that belong to the background frame mask only are filtered out. The motion vectors computed also contain some object motion vectors (because previous background mask is used) which bias the parameter estimation. For this reason, motion vectors that greatly differ from their neighbors (one pixel neighborhood) are rejected. The mean of a 3x3 group of motion vectors is calculated and compared with the motion vector under test. A threshold limit of 1/4 pixel is set as rejection criteria.

Frame warping: The parameters estimated from the above step are used to align the previous/next frame to the current frame. In this process, a new frame is calculated from previous frame by transforming the coordinates of the pixels of the previous frame into a new position. This position is defined by the results of the affine transformation equation given by equation 3.3. Bilinear interpolation given by equation 3.12 is used for computing pixel intensity for non-integer pixel positions for its simplicity (as compared to higher order interpolation methods).

5.3 Statistical change detection

Change detection is used, in each of the cases of the proposed design in section 5.1, to identify MOs. Therefore it is the basic step of the proposed system. Based on the general framework of change detection described in chapter 4, a robust and effective fully automatic change detection system is used for semantic partitioning of MOs from fixed backgrounds or global motion compensated image sequences. This change detection is based on statistical hypothesis testing and with adaptive decision threshold.

The feature space used is the intensity from the YUV color space. The YUV 4:2:0 video format is the format found available for video test sequences. This format consists of a luminance component Y-plane in full resolution and two chrominance components U- and V-planes in half resolution for a single frame of video. From the intensity features of the YUV 4:2:0 format, the luminance component Y of the YUV color space is used. In this format, the Y component carries much information than the other (U and V) components. However, multispectral information can also be used. In Case-2 and Case-3 of the proposed block diagram, intensity normalization given by equation 4.5 is applied for illumination variations before feature analysis. Pixel-wise differencing is used for feature analysis due to its simplicity, which is given by

$$d(x, y) = I_n(x, y) - I_r(x, y) \tag{5.1}$$

where $I_n(x, y)$ is the current frame and $I_r(x, y)$ is the GMOB. In Case-1, the initial background frame I_b, that is obtained prior to segmentation of current frame, is used as reference. Initially, it is set to the first frame of the sequence. In Case-2 and Case-3, the previous and next frames are used, respectively, for forward and backward mask generation.

The classification is done based on adaptive threshold t_α obtained by applying statistical significance testing (similar to the one proposed in [1]) on the result of frame difference as explained in section 4.4 of chapter 4. Due to the camera noise, the result of pixel-wise differencing will not be zero even in areas where there is no change between two frames. This noise is removed during classification by assuming a probability density function for it to compute adaptive threshold by applying statistical hypothesis tests. Under the hypothesis that no change occurs at location *k*, except the camera noise, the difference d_k obeys a zero mean Gaussian distribution $N(0, \sigma)$ with variance σ^2, that is,

$$p(d_k \mid H_0) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{{d_k}^2}{2\sigma^2}\right\} \tag{5.2}$$

Since $p(d_k \mid H_0)$ depends only on the squared ratio $(d_k / \sigma)^2$, the classification is based on this squared ratio. The noise variance is estimated from the background of the frame. A rectangular overlapping window (e.g. 3x3, 5x5) is used for spatial support instead of applying classification based on a single pixel only, in order to make the detection more reliable. Since the ratio (d_k/s) is Gaussian distributed, the sum of the squared ratio $\Delta_i^2 = (d_k / \sigma^2)$ inside the window obeys a Chi-distribution with as many degrees of freedom as there are pixels inside the window. With this distribution known, classification is made by specifying a significance level α and computing a corresponding threshold t_α according to

$$\alpha = \text{Prob}(\Delta_i^2 > t_\alpha \mid H_0) \tag{5.3}$$

For a given significance level α, the threshold t_α is taken from a χ^2-table, for the corresponding value of Δ_i^2. Experiments show that α of the test is not critical and may be varied between 10^{-6} and 10^{-2}. Typical values of alpha were used during testing.

5.4 Post-processing

There are two stages of post-processing applied in the entire system to. The stage **Post-Processing-1** is applied to all of the cases of the proposed system to refine object masks. The stage **Post-Processing-2** is applied to Case-2 and Case-3 only to deal with the problem of holes produced when using consecutive frames for change detection. A new post processing technique is proposed in this work for the second stage of post-processing.

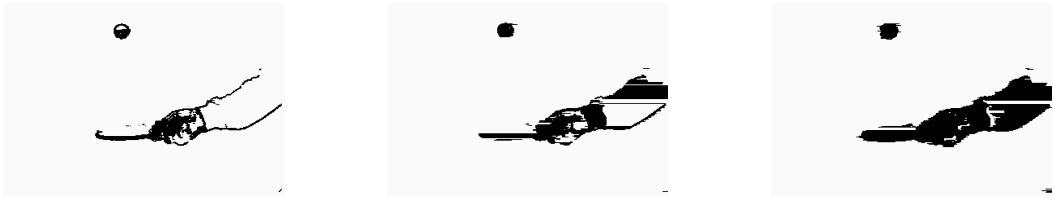
The **Post-processing-1** stage is applied to the segmentation masks produced in both cases to refine the masks and produce final mask, I_{mf} with smooth boundaries. This is applied since the results of change detection obtained usually lack smoothness at boundaries, which is not tolerable in most applications. This stage consists of a series of simple morphological opening and closing operations (presented in Appendix-A).

The second kind of post-processing is the one that is applied in Case-2 and Case-3 of the proposed system. In these cases, different noises have to be reduced to obtain better segmentation results. Three major noise components are removed so far: illumination variation by normalization, uncovered background by frame integration and camera noise by significant testing. These noise components are removed before (or during) the change detection mask is produced. There is also one problem introduced when a previous (or next) frame is used as reference in change detection: large holes are created inside uniform regions of objects segmented, as discussed in section 4.5 of chapter 4. A new post-processing technique is proposed to solve this problem, which is the **Post-processing-2** stage in Figure 5.2. This post-processing technique incorporates the segmentation mask obtained by combining the forward and backward change detection masks and the original current frame I_n. It is explained as follows: since holes are created inside regions with uniform intensities, the difference between neighboring pixels of the original frame inside the regions is small. This small difference together with the neighborhood information is used to decide whether a background pixel is wrongly labeled or not. There are four operations performed on the combined change detection mask (an example shown in Figure 5.4): *forward filling-in*, *backward filling-in, upward filling-in and downward filling-in*. In the forward filling-in, the

mask and current frame are raster scanned from left to right. Pixels that are labeled as objects are maintained. For a pixel of the mask *B(i,j)* that is labeled as background, what is called *right gradient* $g_r(i,j)$ is computed from *I(i,j)* and its right neighbor *I(i,j+1)* of the original current frame as follows:

$$g_r(i,j) = \left| I(i,j) - I(i,j+1) \right| / 256 \tag{5.4}$$

If $g_r(i,j)$ is below some value, and if the left neighbor of the current pixel *B(i,j)* is foreground, then the current pixel *B(i,j)* is labeled as foreground otherwise it is labeled as background. This will fill open spaces to the right, as shown for example in Figure 5.4 b). Experiments showed that a g_r as high as 4% produced better results. Greater values of g_r caused unnecessary regions of background to be included. The same operation is performed on the resulting mask of forward filling-in, in the backward filling-in but from right to left by computing left gradient, and using left neighbor. This will fill open spaces to the left, as shown for example in Figure 5.4 c). These two operations will produce the *horizontal mask*. Similarly, the upward and downward filling-in are applied to the combined mask, the *upward gradient* and *downward gradients* are computed for each pixel using the up and down neighbors respectively, to fill open spaces upward and downward (as shown in Figure 5.4 d) and e)) to produce the *vertical mask*. The final combined mask (Figure 5.4 f)) is then obtained by combining the horizontal and vertical masks by using a logical OR operation.



a) original mask b) result of right fill-in c) horizontal mask

d) upward filling-in e) vertical mask

f) combination of horizontal and vertical masks g)final mask

Figure 5.4 Post-processing steps

CHAPTER 6

IMPLEMENTATION, TESTING AND ANALYSIS OF RESULTS

6.1 Implementation

The proposed system is implemented using C++ on Microsoft Visual C++ IDE. Detail of implementation is presented next.

The general implementation of the system is based on the block diagram presented in section 5.1. The major blocks in the block diagram of the system which are:

- Global motion estimation and compensation
- Change detection and
- Post-processing

These blocks are implemented as classes in the software: `GMEC`, `CD`, and `PP` respectively. These classes and their methods are briefly explained below. For the specific implementation to test the proposed system, only the Y plane of the YUV4:2:0 video format is used. However, the implementation can be easily extended to use the three planes Y, U and V to produce multi-channel segmentation results.

Class – `GMEC`: The class `GMEC` implements the motion estimation and compensation part. In this class, the methods necessary for initializing the ME process with appropriate input frames, for estimating the motion vectors and affine parameters, and for producing and accessing compensated frames are implemented. The following are the methods implemented (sample codes are presented in Appendix-B, section B-1):

`initGMEC()` – is a public method that initializes the global motion estimation. It accepts two `char` buffers for two successive frames and an integer 2-D array for corresponding background mask. It simply extracts the Y components from the two frames and assigns to global 2-D arrays to the class **`previous_y_frame`** and **`current_y_frame`**.

getWarped() – is another public method that is used to get final warped frame – **warped_y**. this function calls the **warpFrame()** method which performs warping the previous frame.

makeBlock() – builds structure of n x m blocks (e.g. 9x11for QCIF sequences) of 16x16 pixels from the current frame for block matching. It takes the current frame array and stores blocks into a **BLOCK struct**.

buildSearchrArea() – this function structures previous frame into **SEARCHAREA structs** from which a current block from **BLOCK struct** in the current frame can be searched for its best match. It also finds candidate motion vectors for a block within its search area, and stores them in a **MV struct**. The **MV struct** encapsulates vertical and horizontal components of motion vectors. The inputs are 2-D frame/pyramid image array of intensities, **height** and **width** of frame/pyramid image and **block_size**.

BMALevel2() - this is the actual block matching function for level-2 pyramid images. It takes **BLOCK** and **SEARCHAREA** structs which are results of **makeBlock()** and **buildSearchArea()**, matches each block of current pyramid image with blocks in search area, finds best match, and stores the corresponding motion vectors in a **MV struct** for propagating to next level. Three best matches are selected for a block and **MV struct**s for each match are stored in **mv1**, **mv2** and **mv3** instances of **MV struct**.

BMA() – this is similar to the above function with the enhancement that it accepts the candidate motion vectors from upper level and performs refinement of these mv's by searching best matches for each block specified by mv's. For level-0, the function finds the final optimal motion vectors by comparing the three candidate mv's.

mEstimate() – this is the major function for the ME. It builds three-level pyramid images from the two input frames: previous and current, initializes **BLOCK**, **SEARCHAREA** and **MV struct**s and calls **makeBlock()** and **buildSearchArea()** methods to compute intermediate and final motion vectors. Intermediate motion vectors are propagated to next level, and final motion vectors are saved.

saveMotionVectors() – saves the final motion vectors.

affineEstmate() – this function performs the 6-parameter affine estimation. First it performs filtering and smoothing of motion vectors and then computes the six parameters by using the gauss elimination method implemented by the function **gauss()**.

warpFrame() – this is used to warp the previous frame according to the affine parameters. The **mEstimate()** and **affineEstimate()** are called by this function, that means the actual ME is performed when this function is called. The bilinear interpolation is performed here. The warped frame is stored in a 2-D array **warped_y** global to the class.

Class – CD: This class implements the proposed change detection algorithm. The methods implemented take the Y components of input frames which are the current frame and motion compensated planes of previous frame or background reference, perform the change detection and produce a binary segmentation mask of the current frame. It offers three public methods for interaction: **initCD(), getObjmask()** and **getBkgmask()**. The following are some of the methods implemented (sample codes shown in Appendix-B, section B-2):

initCD() – initializes the change detection; assigns the Y plane of previous frame passed to it to a 2-D arrays **y_prev** global to the class.

normalizeFrames() – takes two planes, finds mean and variance of the two frames and normalizes the first frame with respect to the second frame.

frameDifference() – first it calls **normalizeFrames()** to perform normalization, then subtracts the plane of previous frame from the plane of the current, and assigns the result to a 2-D array.

produceMask() – this method calls **frameDifference()** and performs classification of the difference image to produce object and background mask arrays **obj_mask** and **bkg_mask**, based on the noise variance estimated by the noiseSd() method.

The methods **getBkgmask()** and **getObjmask()** are implemented to obtain background and object mask respectively.

`Class – PP:` This is the implementation for the post-processing and producing t he final object planes by superimposition. This class takes two change detection object masks (**`prev_mask`** and **`curr_mask`** 2-D arrays) and current frame as input, and produces final object plane. It implements open and close for post-processing, and methods to combine masks and superimpose the mask over the current frame. The following are the methods implemented (sample codes shown in Appendix-B, section B-3):

`combineMasks()` – combines two change detection masks passed to it and produces **`final_mask`** array by using logical AND operation for post-processing.
`openP()` and **`closeP()`** – take a binary mask, apply morphological open and close operations respectively. The output is produced in another binary mask. 3x3 and 5x5 structuring elements are used for open and close.
`superimpose()` – segments those areas from the current frame that belong to the object mask. The result is the final object plane array **`final_obj`**.

The `main()` method

The main method implemented for simulation is explained here. First, it reads file name of the video sequence to be segmented. The file name is expected to be of the form "**.yuv". Then it reads the first frame and the GMOB, or the first three frames of the sequence to initialize the segmentation process. An output file is then created for writing the segmentation result. Next, instances of classes GMEC, CD and PP are created and appropriate methods are called to perform the segmentation process. Finally the segmented object plane is saved to the output file, in the same format as the input file.

6.2 Testing, analysis and evaluation of results

To verify the proposed system and to evaluate the performance, three different standard test video sequences are used, which are obtained from [55]. The three test sequences represent the three cases considered in the design of the proposed system. These sequences are Hall_Monitor and Coastguard in CIF format (288x352 pixels), and Tennis in SIF format

(240x352 pixels). These sequences are stored in a file as sequences of bytes, one byte representing intensity level of a gray or color pixel. The implementation of the system is tested on a Pentium 4 computer with 2.6GHz CPU clock and 256MB memory.

The sequences used for testing are in YUV 4:2:0 format (i.e. full resolution of the luminance Y plane and half resolution of chrominance U and V planes) and contain different dynamics. For viewing the input video sequences frame by frame and the segmentation output video object planes, the raw video player "YUV Player" from [56] is used. The results show the Y planes (i.e. the luminance components, without the color planes) of the segmented sequences, since only this plane is used in the specific implementation of the system. Subjective evaluation is done, which is a method of evaluation based on visual observation of segmentation results. In this method of evaluation the results are observed for any artifacts, edge smoothness, loss of details and inclusion of background and the observer judges the results based on these observations.

6.3.1 Analysis and subjective evaluation for Case-1

The first sequence used to test the system was the Hall_Monitor sequence which is an indoor surveillance video sequence and is used to test Case-1 of the proposed system. In this sequence, an initial background frame can be easily obtained, which is the first frame (shown in Figure 6.1). The performance of the change detection was analyzed in this case, by using different spatial window sizes, and with and without post-processing. Three most commonly used window sizes, 3x3, 5x5 and 7x7 are used. For classification, a significance level α=0.001, which is between 10^{-2} and 10^{-6} selected. The thresholds that correspond to this significance level for the above window sizes are 27.8, 52.6 and 85.3 respectively. For post-processing, a sequence of close and open operations is used. Sample results of segmentation for this sequence are shown in Figure 6.2 for frame numbers 38, 50 and 150.

Figure 6.1 The first frame (frame number 0) of Hall_Monitor video sequence



(a) (b) (c)



(a_1) (b_1) (c_1)



(a_2) (b_2) (c_2)

(a_3) (b_3) (c_3)

(a_4) (b_4) (c_4)

(a_5) (b_5) (c_5)

Figure 6.2 Segmentation results for the Hall_Monitor sequence: (a), (b), (c) original frames 38, 50 and 150 respectively; (a_1) – (c_1) segmented SVOs for window size 3x3 without post-processing; (a_2) – (c_2) for window size 5x5 without post-processing (a_3) – (c_3) for window size after applying one pair of post-processing; (a_4) – (c_4) for window size 7x7 without post-processing; (a_5) – (c_5) for window size after applying one pair of post-processing.

The above results show that the adaptive model-based change detection used successfully segmented the SVOs only. Negligible amount of noise (part of background) is detected with the SVOs. Even this is easily eliminated by post-processing. The spatial window size and the post-processing stage are observed to influence the results of segmentation. As shown in

Figure 6.2 (a_1) – (c_1), window size of 3x3 did not perform well, since parts of the object are segmented as background, which is difficult to correct by applying post-processing. Better results were obtained using larger window size of 5x5 as shown in (a_2) – (c_2). These results were refined using post processing as shown in (a_3) – (c_3), which show more accurate boundaries and regions of the semantic objects. The 7x7 window size also produced good results as shown in (a_4) – (c_4). As the window size is increased, visible spurious regions (artifacts) are detected along the object's boundary because of the windowing effect. This effect is removed by post-processing as shown in (a_5) – (c_5). In these results, the shadow of the object was detected, since no special shadow removing technique is included.

6.3.2 Analysis and subjective evaluation for Case- 2

The next sequence tested was the Tennis sequence, which shows the movements of hand of a tennis player and a ball. In this sequence, there is no initial background frame that can be obtained without any semantic objects. This sequence is therefore used to test Case-2 of the proposed system. Sample results of segmentation for this sequence are shown in Figure 6.3.

In this sequence also, the change detection detected the MO only, and negligible amounts noise regions (near the edges of the ball) from background. In this sequence, the larger portion of the regions consisting of the MOs (the hand and the racket) is a uniform region. The change detection therefore does not detect the centers of these regions, only the changed regions near the edges are detected as shown for example in Figure 5.4 a). However the proposed post-processing stage (Post-processing-2 of the block diagram of the system) effectively filled the open regions formed as shown in the Figure 6.3. For this sequence also, the spatial window size and the post-processing stage influence the results of segmentation. The results using the two window sizes, 5x5 and 7x7 are presented in Figure 6.3, after applying one and two pairs of post-processing. According to our observation, both window sizes produced almost visually equivalent results. However, there is a small difference between using one and two pairs of open close post-processing. The use of two pair open close operation (Figure 6.3 (a_2) – (c_2) and (a_4) – (c_4)) produced better results than the use of one pair open close operation (Figure 6.3 (a_1) – (c_1) and (a_3) – (c_3)) especially when smoothness of boundaries is observed. Part of the hand was not detected in the results for frame 03 since only the portion of the hand that is detected is moving in the original sequence.

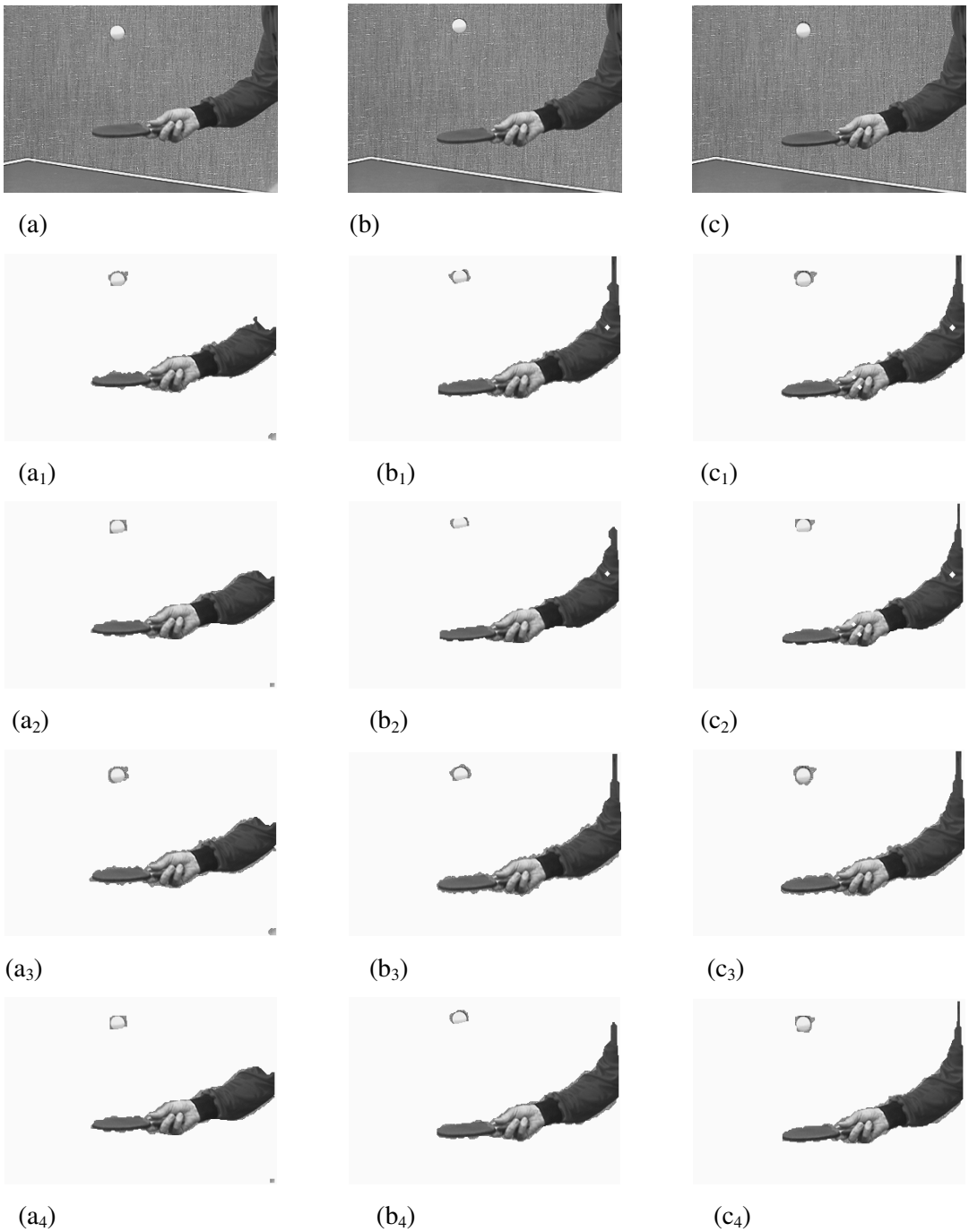
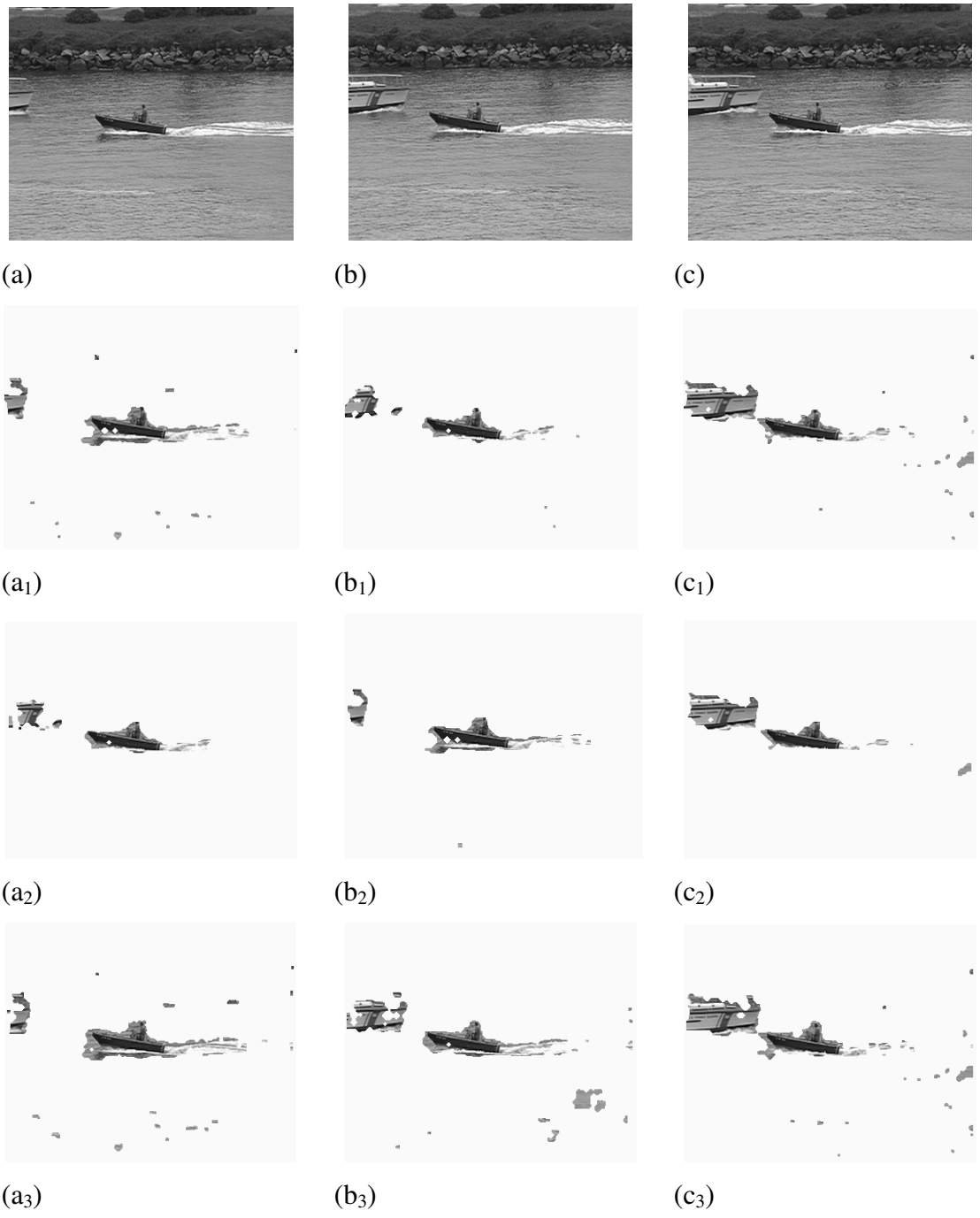


Figure 6.3 Segmentation results for the Tennis sequence: (a), (b) and (c) original frames 03, 05 and 06 respectively; and segmentation results of (a_1) – (c_1) window size 5x5 after applying one pair of open-close operation; (a_2) – (c_2) window size 5x5 after applying two pair of open-close operation; (a_3) – (c_3) window size 7x7 after applying one pair of open-close operation; (a_2) – (c_2) window size 7x7 after applying two pair of open-close operation**.**

Analysis and subjective evaluation for Case-3

The third sequence used to test the system is the Coastguard sequence. This sequence consists of background movement as well as object movement. This is used to test Case-3 of the proposed system. Some of the results of segmentation are shown in Figure 6.4.

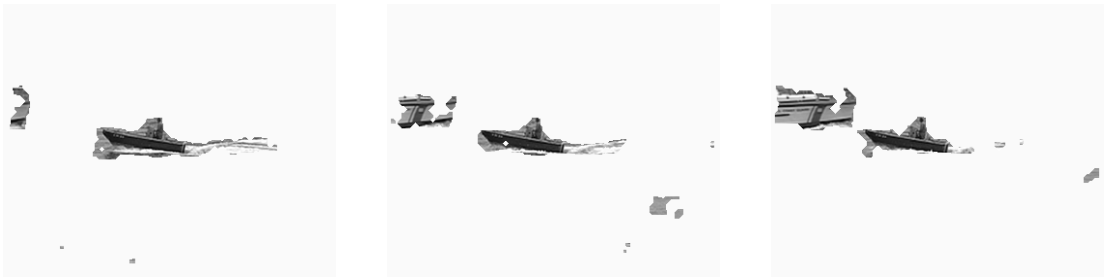


(a) (b) (c)

(a_1) (b_1) (c_1)

(a_2) (b_2) (c_2)

(a_3) (b_3) (c_3)



(a_3) (b_3) (c_3)

Figure 6.4 Results of the Coastguard sequence: (a_1), (a_2) and (a_3) original frames 04, 17 and 21 respectively; and segmentation results of (a_1) – (c_1) window size 5x5 after applying one pair of open-close operation; (a_2) – (c_2) window size 5x5 after applying two pair of open-close operation; (a_3) – (c_3) window size 7x7 after applying one pair of open-close operation; (a_2) – (c_2) window size 7x7 after applying two pair of open-close operation**.**

This sequence is very difficult to segment, since it contains different motions of background and object. The MO is the small boat. Then suddenly appears a larger boat. The water surface also undergoes a random movement (waves) in addition to the motion of the background. There is also trailing movement of the water surface left behind the moving boat which should not be detected. This is very difficult to totally avoid, since it produces change. In spite of these difficulties, the system successfully segmented the MO, although the boundaries are not perfect. Since the background motion estimation and compensation technique selected aligned the frames exactly, no background movements were detected. Some noise regions which parts of background are detected due to movement of the water surface. This is reduced by post-processing.

In this case also, the two window sizes and one and two pairs of open close post-processing operations are used for analysis. As shown in Figures 6.4 (a_1) – (c_1) and (a_3) – (c_3), more spurious regions are detected that belong to the background. One pair of open close operations did not remove all the regions. Boundaries of objects are not so clear. This is because the water surface is irregular and there is no clear boundary between the object and the water surface. The result is more improved in using two pairs of open close post-processing as shown in Figures 6.4 (a_2) – (c_2) and (a_4) – (c_4).

6.3.3 Subjective evaluation from different observers

Subjective evaluation by different observers is used to assess the qualities of segmentation results, and to fix the values of the window size to be used and the number of open close operations to be applied. Subjective evaluation is a qualitative human judgment by observing the segmentation results. A group of five people (MSc students in Electrical and computer Engineering Department, Addis Ababa University) are made to observe the segmentation results. The four rows of results from Case-1 shown in Figure 6.2 (a_2) – (c_2) to (a_5) – (c_5) using both window sizes and, without one pair of open close post-processing (i.e. a total of 20 observations) were presented for observation. Out of the 20 observations, 60% (three observers) showed that the results of both window sizes and with one pair of open close post-processing are almost identical and visually pleasing and with smooth boundaries. The four rows results from Case-2 shown in Figure 6.3 (a_1) – (c_1) to (a_4) – (c_4) using both window sizes and after applying one and two pairs of open close post-processing (i.e. a total of 20 observations) are presented for observation. 80% of observations (four observers) showed that the results of both window sizes are almost identical and 80% of observations showed that the results of two pairs of open close operations are better. From these two observations, it can be noted that both window sizes produced comparable results. However, since larger window sizes increase computation intensity, 5x5 window size is preferable. The results of a pair of open close operations for Case-1 and the results of two pairs of open close operations for Case-2 produced better results. The results of Case-3 are not presented for evaluation by the group since the difference is clearly visible. From the results it can be easily identified that window size of 5x5 with two pairs of produced better results and also, only small amounts of parts of background is detected.

6.3.4 Comparison of the different cases

In Case-1, only three stages are used for segmentation and very good results are obtained. This is because there is an initial background GMOB is used. The change detection stage effectively detected regions of objects, and simple open close post-processing produced visually pleasing results in this case. Video sequences with initial background reference can

benefit greatly from this simplified segmentation process. In Case-2 and Case-3, there is no initial background assumed, and additional stages in the forward and backward process are required to deal with this. The change detection was not able to detect uniform regions of objects and so, the new post-processing stage is introduced to improve the results of segmentation. Furthermore, in Case-3, additional stages are added to deal with video sequences containing camera movements.

Comparison with other systems

Comparing the proposed system with other ASsystems is found to be very difficult. This is because different systems used different input sequences for testing and analysis. Also the objective and applications of these systems are different. However, a few segmentation results are found and presented in Figure 6.5 just for visual comparison only. Figure 6.5 (a) shows segmentation result for frame 150 of the Hall_Monitor sequence obtained from [10], which is a change detection-based technique. Figure 6.5 (b) and (c) show the result of the proposed system for the same frame. The background of result (b) is made black for the purpose of comparison, since the result from [10] is presented that way. However, the same result is presented with white background. Although [10] uses change detection, it constructs background GMOB for comparison with the current frame by integrating a number of previous frame differences. This complicates the segmentation process and requires more processing time to maintain the background. However, the proposed system uses the advantage of existing background GMOB and performs simple change detection. Based on our observation, the proposed system detected more parts of objects (the bag and hand of the person on the right). But both results have comparable boundary smoothness and visual quality. However, this is only based on rough observation, since the image presented for comparison is blurred, and is difficult to observe exact boundaries and parts of objects.

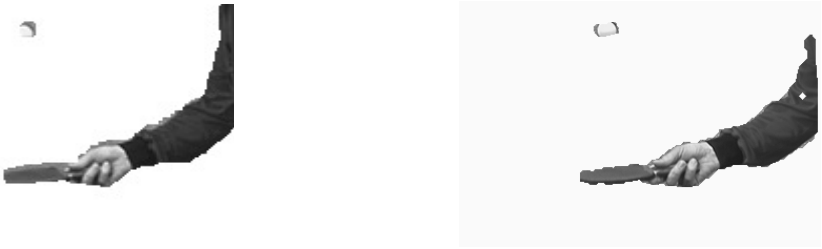
Figure 6.5 (d) shows result for frame 05 of the Tennis sequence from [52] which is a GMOB based technique and (e) shows result of the proposed system for the same frame obtained using 5x5 window size and after two open close post-processing operations.

The system in [52] performs ME for detecting areas of MOs and to obtain initial model of objects. Then, spatial segmentation of frames into regions is performed and the regions are

matched with the initial model to obtain final objects. However, both ME and GMOB are computationally expensive processes (as explained in section 2.1 of chapter 2) which cannot be compared with the simple change detection used in the proposed system.

This comparison shows that the proposed system produced more accurate boundaries. However, parts of the hand are not detected; this is because that part of the hand is not moving.

(a) result from [10] (b) result of proposed system (c) result of proposed system



(d) result from [52] (e) result of proposed system

Figure 6.5 Comparison of segmentation results of proposed system with similar techniques

Subjective comparison of results from different observers

The same group of people used above is made to compare segmentation results shown in Figure 6.5 also. For the comparison of results of the Hall_Monitor sequence (Figure 6.5 (a) and (b)) out of five people, 80% reported that the result of the proposed system is more visually pleasing, and with better smoothness of boundaries. For the results of the Tennis

sequence, 100% reported that the result of the proposed system is smoother and more visually pleasing.

6.3.5 Computational complexity analysis

To examine the time efficiency, computation times are determined for the specific implementation and for the particular machine used for testing. Average values of computation times over the frames tested for each sequence are given in Table 6.1 below. The computation times for the algorithms used in the three main blocks of the system, change detection, global motion estimation and compensation and post-processing (both filling-in and open close) are also presented in Table 6.2.

Table 6.1 Computation times for the tested sequences

Sequence	Frame size	Computation time (msec)
Hall_Monitor (case 1)	288x352	188
Tennis (case 2)	240x352	510
Coastguard (Case 3)	288x352	3625

Table 6.2 Computation times for the main algorithms

Algorithm	Computation time (msec)
Change detection	156
Global motion estimation and compensation	1320
Post-processing (filling-in + open close)	180

The computation time for the Hall_Monitor sequence is near to that of the time for the change detection algorithm, since only open close processing is applied, as described in Case-1 of the design. For the Tennis and Coastguard sequences, computation times are more than two times the respective algorithms used, since the algorithms are executed twice (during forward and backward mask generation) and also since both post-processing stages are applied. It was tried to compare the computational time results stated above with other segmentation systems. However, the other systems did not provide computational time

results. However, it can be observed that the computation times are much smaller than those expected in human assisted segmentation systems.

The analysis and comparison of the proposed system presented above is based on subjective basis. There is also another method of performance evaluation which is objective evaluation. However, this evaluation method requires comparison of segmentation masks with ground-truth data, which is hand segmented sequence of frames. In this method of evaluation, the error rate of segmentation is determined by calculating the number of erroneous pixels added to the background and subtracted from objects. Since ground-truth data was not obtained, the system was not evaluated on this basis.

Chapter 7
CONCLUSION AND FUTURE WORK

7.1 Conclusions

Automatic segmentation of SVOs is required by many content-based applications to fulfill their demands and to avoid the burden of human assistance in segmentation. This thesis has concentrated on fully automatic SVO segmentation system. The design considered video sequences that contain both static and moving backgrounds. Different cases have been considered in designing the system to make it more flexible.

The three cases considered in the design of the proposed system have been tested and the results have been evaluated. The proposed system used statistical model based change detection for automatic SVO segmentation in all the 3 cases considered. The change detection uses adaptive threshold computation based on camera noise. The adaptive threshold is used to automatically label pixels changed or unchanged. For applications where there is initial background GMOB, the change detection algorithm used the first frame as reference. For sequences where initial background frame cannot be obtained, the proposed system used three consecutive frames and effectively combined two change detection results to reduce the effect of uncovered background. For sequences with moving backgrounds, a simplified block-based motion estimation and compensation is used. In each of the cases considered in the design, analysis of test results showed that the system successfully segmented SVOs from the sequences tested. This shows that the selected techniques worked well and the intended fully AS system is achieved.

Evaluation of the results of Case-1 showed that among the three window sizes tested, 5x5 and 7x7 produced better and comparable results in terms of visual quality and boundary smoothness. 60% of the visual observations made also showed this. This visual observation also showed that both window sizes produced visually pleasing results with smooth boundary

after one pair of open close post-processing. This, together with the smaller computational times observed, shows that computation intensity of the change detection can be reduced by using the 5x5 window size without affecting visual quality of segmentation with smaller processing times in this case as compared to other cases.

In general, it can be concluded from the results of this case that when there is an initial background GMOB, only few steps and simplified technique (change detection and simple open close post-processing) can be used in the segmentation process without affecting quality of segmentation results. One problem observed in this case, is that shadows of objects are detected and this requires that explicit shadow detection mechanism should be incorporated.

In Case-2 and Case-3 of the proposed system, evaluation of the results showed that the new post-processing technique applied successfully solved the problem of holes inside uniform regions. In these cases also, the results are observed to be influenced by the window size and the number of open close operations applied. It was also observed that the same optimal values obtained in Case-1 for window size produced better results, although the number of open close operations applied increased by one pair. This produces only a small increase in computation, but improved the results greatly. In these cases, a problem that is observed is that parts of objects that stop moving are not detected. This is one limitation of the proposed system.

The subjective comparison of results with other systems for Case-1 and Case-2 showed that larger percentage of observations reported better visual quality and smooth boundaries in the results of the proposed system. Although this comparison is rough, it shows that an improved system is achieved. The processing times indicate reduced delay during segmentation (as compared to human assisted systems), although this does not fulfill real time requirements. This reduced delay is the result of selecting algorithms with reduced computational intensity.

In general, it can be concluded that an effective and flexible fully automatic SVO segmented system for various types of video sequences is achieved in this thesis by selecting effective and efficient techniques with reduced computation complexity, and promising results have

been obtained in terms of visual quality of segmentation results. The main contributions are simplified but effective approach to solution of AS problem and a new post-processing technique to solve the problem of holes created inside uniform regions of objects during change detection.

7.2 Future works

Although evaluation of the test results showed that a successful segmentation system is achieved, further improvements and extensions can be made, in addition to dealing with the limitations observed. In general, the following points can be recommended as improvement and future work:

1. Combining two change detection masks increases the delay by almost two fold in case 2 and 3. This can be avoided by using one change detection mask, but by applying more complex region-based post-processing.
2. The six parameter affine motion model used in global motion estimation works under the assumption of planar scenes. For complex perspective scene distortions, the model can be extended to more complex models. However, this requires careful attention since ME is the most computation intensive part of the proposed system, as the processing times show.
3. Region-based segmentation can be integrated with the system after applying change detection to deal with the problem that results when parts of objects stop moving. Another solution to this may be to collect previous history of the objects detected by incorporating memory element.
4. Some applications require specific objects to be independently extracted. The system can be extended to complete extraction of semantic objects by adding object tracking mechanism.
5. At last, the implementation of the system can be improved by using more efficient data structures to reduce delay and to achieve real-time requirements.

REFERENCES

1. Aach T. et al. (1993) "Statistical Model-based Change Detection in Moving Video." *Elsevier Signal Processing*, vol. 31, no. 2, pp. 165-180.
2. Barjatya A. (2004) "Block Matching Algorithms for Motion Estimation." DIP 6620, www.ndt.net/article/panndt2007/papers/14.pdf (3/3/2007).
3. Bowley C. (2004) "Global Motion Approximation." *BBC R&D*, http://dirac.sourceforge.net/documentation/algorithm/global_motion/global_motion.pdf (3/4/2007)
4. Castleman R. (1996) "Digital Image Processing." *Prentice Hall*, New Jersy, pp. 470-475.
5. Cavallaro A. (2002) "From Visual Information to Knowledge: Semantic Video Object Segmentation, Tracking and Description." *PhD Thesis*, Swiss Federal Institute of Technology, Switzerland.
6. Cavallaro A and Ebrahimi T. (2004) "Interaction Between High-Level Image Analysis for Semantic Video Object Extraction." *EURASIP,* vol. 6, pp. 786-797.
7. Chen T-H, Liau H-S and Chiou Y-C. (2005) "An Efficient Video Object Segmentation Algorithm Based on Change Detection and Background Updating." *Kun Shan University, National Computer Symposium*, MIA1-2 (MI14)
8. Chen Y-S, Hung Y-P and Fuh C-S. (2001) "Fast Block Matching Algorithms Based on the Winner-Update Strategy." *IEEE Transactions on Image Processing,* vol. 10, no. 8, pp. 1212-1222.
9. Chen Y, Wang Y and Lu Y. (1998) "A New Fast Motion Estimation Algorithm – Literature Survey." http://users.ece.utexas.edu/~bevans/courses/ee381k/projects/fall98/chen-lu-wang/literatureSurvey.pdf, (13/4/2007)
10. Chien S-Y, Ma S-Y and Chen L-G. (2002) "Efficient Moving Object Segmentation Algorithm Using Background Registration Technique." *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 7, pp. 577-586.
11. Dante A and Brookes M. (2003) "Precise Real-Time Outlier Removal from Motion Vector Fields for 3-D Reconstruction." *IEEE Transactions on Image Processing*, vol. 1, pp. 393-396.

12. Drucan E and Ebrahmi T. (2000) “Robust and Illumination Invariant Change Detection Based on Linear Dependence for Surveillance Application.” *In Proc. EUSIPCO 2000*, pp. 1041-1044, Tampere, Finland.
13. Farin D, Peter N H and Effelsberg W. (2004) “Video Object Segmentation Using Multi-Sprite Background Subtraction.” *in IEEE International conference on Multimedia and Expo, ICME 2004*, vol. 1, pp. 343-346.
14. Farin D and With H N. (2005) “Evaluation of a Feature-Based Global-Motion Estimation System.” *In Proc. SPIE*, vol. 5960, pp. 1331-1342.
15. Gachter S. (2001) Motion Detection as an Application for the Omnidirectional Camera. Center for Machine Perception, Department of Cybernetics, Faculty of Electrical Engineering Czech Technical University ftp://cmp.felk.cvut.cz/pub/cmp/articles/pajdla/Gaechter-TR-2001-07.pdf
16. Gatica PD, Gu D, Sun MT. (2001) “Semantic Video Object Extraction Using Four-Band Watershed and Partition Lattice Operators.” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 5, pp. 603-618.
17. Guo J, Kim J and Kuo CCJ. (1999) “Fast and Adaptive Semantic Object Extraction from Video.” *IEEE Transactions on Signals, Systems and Computers*, vol. 2, pp. 1417-1421.
18. Gupta G. and Chakrabarti C. (1995) “Architectures for Hierarchical and Other Block Matching Algorithms.” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, pp 477-489.
19. Hoynck M. et al. (2004) “Robust Object Region detection in Natural Video Using Motion Estimation and Region-Based Diffusion.” *In Proceedings of the International Picture Cding Symposium (PCS2004)*, San-Francisco, USA.
20. Hoynck M, Unger M and Ohm J-R. (2004) “Robust Object Region Detection in Natural Video using Motion Estimation and Region-Based Diffusion.” *Institute of Communications Engineering (IENT)* RWTH Aachen University D-52056 Aachen, Germany
21. Jang S-W, Pomplun M, Kim G-Y and Choi H. (2005) “Adaptive Robust Estimation of Affine Parameters from Block Motion Vectors.” *ELSEVIER* vol. 23, pp. 1250-1263.
22. Kim C and Hwang J-N. (2002) “Fast and Automatic Video Object Segmentation and Tracking for Content-Based Applications.” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 2, pp. 603-618.

23. Lee C-H and Chen L-H. A (1997) "Fast Motion Estimation Algorithm Based on the Block Sum Pyramid." *IEEE Transactions on Image Processing*, vol. 6, no. 11 pp. 1587-1591.
24. Liau H-S and Chiou Y-C. (2005) "An Efficient Video Object Segmentation Algorithm Based on Change Detection and Background Updating." *Kun Shan University, National Computer Symposium*, MIA1-2 (MI14)
25. Long F, Feng D, Hanchuan P and Siu WC (2001) "Extracting Semantic Video Objects." *IEEE Transactions on Computer Graphics and Applications*, Jan/Feb, pp. 48-54.
26. Lou J. et al. (2002) "An Illumination Invariant Change Detection Algorithm." *The 5th Asian Conference on Computer Vision*, 23-25 January 2002, Melbourne, Australia.
27. Lu Z and Pearlman WA. (2000) "Semi-Automatic Semantic Object Extraction for Video Coding." *IEEE Transactions on Image Processing*, vol. 1, pp. 304-307.
28. Marcenaro L, Gera G and Regazzoni C. (2000) "Adaptive Change Detection Approach for Object Detection in Outdoor Scenes under Variable Speed Illumination Changes." *European Signal Processing conference 2000 (EURASIP2000)*, Tampere (SF), Finland
29. Martínez J M. (2004) "MPEG-7 Overview (version 10)." *ISO/IEC* JTC1/SC29/WG11 N4668.
30. Mezaris Vs, Kompatsiaris I and Strintxis M G. (2004) "Video Object Segmentation Using Bayes-Based Temporal Tracking and Trajectory-Based Region Merging." *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 14, no. 16, pp. 782-794.
31. Nam K. M., Kim J.-S., Park R-H and Shim Y. S. (1995) "A Fast Hierarchical Motion Vector Estimation Algorithm Using Mean Pyramid." *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 4, pp. 344-351.
32. Oh JH, Sankuratri P and Tavanapong W. (2003) "Efficient Measuring of Various Motions in MPEG Videos." *In Proc. the 9th IASTED International Conference on Signal and Image Processing (SIP 2003),* Honolulu, Hawaii, USA.
33. Olsen S I. (1993) "Noise Variance Estimation in Images." *In Proc. 8th SCIA*, May 25-28, Norway.
34. Pan J, Li S, Zhang Y-Q. (2000) "Automatic Extraction of Moving Objects Using Multiple Features and Multiple Frames." *In Proc. IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, vol. 1, pp. 35-39.

35. Radke R J et al. (2005) “Image Change Detection Algorithms: A Systematic Survey.” *IEEE Transactions on Image Processing*, vol. 14, no. 3 pp. 294-307.
36. Rhemann C. (2005) “Region-Based Optical Flow Estimation with Treatment of Occlusions.” *MSc Thesis*, Vienna University of Technology.
37. Rob K. (2002) “MPEG-4 Overview - (V.21 – Jeju Version).” *ISO/IEC*, JTC1/SC29/WG11 N4668.
38. Roerdink J B T M and Meijster A. (2001) “The Watershed Transform.” *Fundamenta Informatica, (IOS Press)*, vol. 41, pp. 187-228.
39. Rosin P L. (1998) “Thresholding for Change Detection.” *ICCV*, pp. 274-279.
40. Rosin P L and Ellis T. (1995) “Image Difference Threshold Strategies and Shadow Detection.” *In Proc. the 1995 British Conference on Machine Vision*, vol. 1, pp. 347-356.
41. Saykol E, Gudukbay U and Ulusoy O. (2001) “A Semi-Automatic Object Extraction Tool for Querying in Multimedia Databases.” *In proceedings of 7th Workshop on Multimedia Information Systems*, November 2001, pp. 11-20, Capri, Italy
42. Sifakis E, Grinias I and Tziritas G. (2002) “Video Segmentation Using Fast Marching and Region Growing Algorithms.” *EURASIP Journal on Applied Signal Processing*, vol. 2002, pp. 379-388.
43. Sifakis E and Tziritas G. (1999) “Fast Marching to Moving Object Location.” *In proceedings of the Second International Conference on Scale-Space Theories in Computer Vision*, September 26-27 1999, vol. 1682, pp 447-452.
44. Sirtkaya S. (2004) “Moving Object Detection in 2D and 3D Scenes.” *MSc Thesis*, Middle East technical University.
45. Smolic A and Ohm J–R. (2000) “Robust Global Motion Estimation Using a Simplified M-Estimator Approach.” *In Proc. International Conference on Image Processing*, vol. 1, pp. 868-871.
46. Su C-H, Hang H-M and Lin D W. (1999) “Global Motion Parameter Extraction and Deformable Block Motion Estimation.” *In IEICE Transactions on Information and Systems*, vol. E82-D, no. 8, pp. 1210-1218.
47. Tab F A. (2002) “Multiresolution Scalable Image and Video Segmentation.” *PhD Thesis*, School of Electrical and Computer and Telecommunications Engineering, University of Wollongong.

48. Thakoor N, Gao J and Chen H. (2004) "Automatic Object Detection in Video Sequences with Camera in Motion." *In Proc. Advanced Concepts for Intelligent Vision Systems*, 2004.
49. Tsaig Y and Averbuch A. (2002) "Automatic Segmentation of Moving Objects in Video Sequences: A Region Labeling Approach." *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 7, pp. 597-612.
50. Tsaig Y. (2002) "Automatic Segmentation of Moving Objects in Video Sequences." *Department of Computer Science School of Mathematical Sciences Tel-Aviv University*, Tel-Aviv 69978, Israel
51. Turaga D, Alkanhal M. (1998) "Search Algorithms for Block-Matching in Motion Estimation." http://www.ece.cmu.edu/~ee899/project/deepak_mid.htm (16/04/2007)
52. Wei WEI. (2003) "Automatic Video Object Segmentation for MPEG-4." *SPIE International Conference on Visual Communications and Image Processing*, 2003, pp.9-19.
53. Zahariadis T. and Kalivas D. (1996) "A Spiral Search Algorithm for Fast Estimation of Block Motion Vectors." *In Proc. the Eighth European Signal Processing Conference (EUSIPCO 96), Signal processing VIII, theories and applications*, vol.2, pp. 1079-1082.
54. Zhang D and Lu G. (2001) "Segmentation of Moving Objects in Image Sequence: A Review." *Circuits, Systems and Signal Processing*, vol.20, pp. 143-183
55. Center for Image Processing Research, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, http://www.cipr.rpi.edu/resource/sequences/sif.html (12/06/2007)
56. Troy, NY 12180 University of Bath, http://www.bath.ac.uk/elec-eng/research/sipg/resource/YuvFilePlayer/YuvFilePlayer.zip (18/06/2007)

Appendix – A

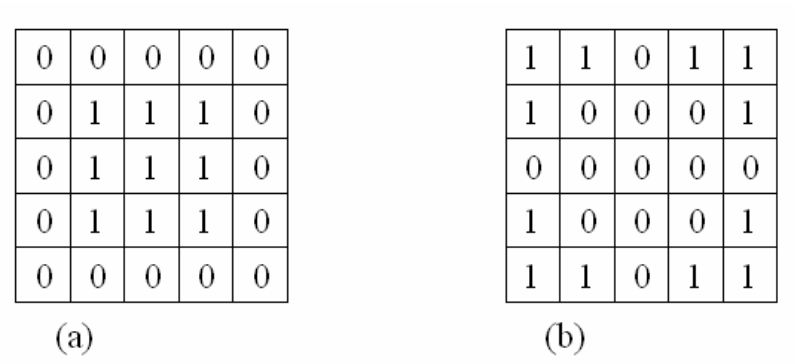
MORPHOLOGICAL OPERATIONS

Mathematical morphology is a set of binary image processing operations developed from a set-theoretic approach [4]. The basic operations are simple, but can be concatenated to produce more complex effects. In general case, morphological image processing operates by passing a structuring element over the image in an activity similar to convolution. The structuring element can be of any size and can contain any complement of 0's and 1's. At each pixel position, a specified operation is performed between the structuring element and underlying binary image. The binary result of that logical operation is stored in the output image at that pixel position. The effect created depends on size and content of the structuring element and on nature of logical operation.

The basic operations are **erosion** and **dilation**. Erosion is the process of eliminating all boundary points from an object, leaving the object smaller in area by one pixel all around its perimeter. Objects no more than two pixels in any direction are eliminated. A boundary point is a pixel that is located inside an object, but that has at least one neighbor outside the object. Erosion is useful for removing from a segmented image objects that are too small to be of interest. It is given by the following set operation:

$$E = B \otimes S = \left\{ x, y \mid S_{x,y} \subseteq B \right\}$$

Where B is the original image, S is called the *structuring element*. That is, the binary image *E* that results from eroding *B* by *S* is the set of points *(x,y)* that is completely contained within B. the structuring element S determines the precise details of the effect of the operator on the image and consists of a pattern specified as the coordinates of a number of discrete points relative to some origin. Normally Cartesian coordinates are used and so a convenient way of representing the element is as a small image on a rectangular grid. Figure A-1 shows structuring elements used for open close post-processing in this thesis. When morphological operation is carried out, points within the structuring element are compared with the underlying image pixel values



0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

(a)

1	1	0	1	1
1	0	0	0	1
0	0	0	0	0
1	0	0	0	1
1	1	0	1	1

(b)

Figure A-1 Structuring elements used in open close operations: a) structuring element for dilation b) structuring element for erosion.

Dilation is the process of incorporating into the object all background points that touch it, leaving it larger in area by that amount. It is useful for filling holes in segmented objects. It is given by

$$D = B \oplus S = \left\{ x, y \mid S_{x,y} \cap B \neq \phi \right\}$$

That is, the binary image that results from dilating B by S is the set of points (x,y) such that if S is translated so that its origin is located at (x,y), then its intersection with B is not empty.

Opening and closing

Opening – is the process of erosion followed by dilation. It has the effect of eliminating small and thin objects breaking objects at thin points, and generally smoothing boundaries of larger objects without significantly changing their area

$$B \circ S = (B \otimes S) \oplus S$$

Closing – is the process of dilation followed by erosion. It has the effect of filling small and thin holes in objects connecting nearby objects, and generally smoothing boundaries of objects without significantly changing their area

$$B \bullet S = (B \oplus S) \otimes S$$

Often, when noisy images are generated by thresholding, boundaries are quite ragged, objects have false holes, and background is peppered with small noise objects. Successive openings and/or closings can improve the situation markedly. Sometimes several iterations of erosion followed by dilation produce desired effect.

Appendix B

SAMPLE SOURCE CODES

B-1 GMEC Class

```
//*************************************************************************
 /*
       Header file for Hierarchical Mean Pyramid based
       Global motion estimation and compensation
       Written by: Eneyachew Tamir
       MSc project
       Addis Ababa University
       Aug 2007
  */
//*************************************************************************

#ifndef SVO_GMEC_H
#define SVO_GMEC_H

#include "svo_util.h"

class GMEC {
public:
       GMEC();
       ~GMEC();

       //methods
       void initGMEC(unsigned char *frame1, unsigned char *frame2,
                     int bkg[][FRAME_WIDTH], int frame_no, int h,int w);
       void getWarped(float y[][FRAME_WIDTH);

private:
       void convert(unsigned char *buf1, float y[][FRAME_WIDTH], int h, int w);
       void makeBlock(float a[][FRAME_WIDTH],BLOCK *b,int h,int w,int block_size);

       void buildSearchArea(float pyramid[][FRAME_WIDTH], SEARCHAREA *SA1,
                            MV *mv, int  h, int w, int block_size);
       void BMALevel2(BLOCK *b, SEARCHAREA *SA, MV *mv1, MV *mv2, MV *mv3,
                      int block_size);
       void BMA(BLOCK *b, SEARCHAREA *SA1, SEARCHAREA *SA2,SEARCHAREA *SA3,
                MV *mv, MV *mv1, MV *mv2, MV *mv3, int block_size);
       void saveMotionVectors();
       void affineEstmate();
       void newAffine();
       void mEstimate();
       void warpFrame();

       //attributes
       float  (*previous_y_frame)[FRAME_WIDTH],
              (*current_y_frame)[FRAME_WIDTH],
              (*warped_y_frame)[FRAME_WIDTH],

       int    (*bkg_mask)[FRAME_WIDTH];

       //original frame dimensions
       int  orig_frame_height, orig_frame_width;
       int  frame_no;
       int block_size_0;


       //final motion vectors
```

```
	MV *mv_final;

	//original frame blocks
	BLOCK *orig_block;

	//affine parameter vector
	double A[6];
};
#endif //SVO_GMEC_H

//********************** end of GMEC class header ************************

//****************************************************************************
 /*
	Implementation for Hierarchical Mean Pyramid based
	Global motion estimation and compensation
	Written by: Eneyachew Tamir
	MSc project
	Addis Ababa University
	Aug 2007
  */
//****************************************************************************

#include <iostream.h>
#include <fstream.h>
#include <cstdlib>
#include <math.h>
#include "svo_gmec.h"

//*****constructor implementation for initialization and mem allocation****
GMEC::GMEC()
{
	register int i;

	//allocate mem for prev, curr and warped frames, and
	//for background bask
	previous_y_frame = new float[FRAME_HEIGHT][FRAME_WIDTH];
	current_y_frame  = new float[FRAME_HEIGHT][FRAME_WIDTH];
	warped_y_frame   = new float[FRAME_HEIGHT][FRAME_WIDTH];
	bkg_mask = new int[FRAME_HEIGHT][FRAME_WIDTH];

	//allocate mem for final motion vectors
	mv_final   = new MV[NUM_BLOCKS];
	orig_block = new BLOCK[NUM_BLOCKS];

	orig_frame_height = 0;
	orig_frame_width  = 0;
	block_size_0      = 16;

	//initialize mv to zero
	for(i=0; i<NUM_BLOCKS; i++){
		mv_final[i].mvx = 0;
		mv_final[i].mvy = 0;
		orig_block[i].bvalue = new float[BLOCK_AREA];
	}

	//initialize the parameter vector
	for(i=0; i<6; i++){
		A[i] = 0;
	}
}
//****************************************************************************

//******************** function to initialize GMEC ************************
//GMEC is initialized by two frames: previous and current
void GMEC::initGMEC(unsigned char *img1, unsigned char *img2,
                    int bkg[][FRAME_WIDTH], int n, int h, int w)
{
	//convert char buffers to arrays
	convert(img1,previous_y_frame, h, w);
	convert(img2,current_y_frame, h, w);
```

```
        orig_frame_height = h;
        orig_frame_width = w;
        frame_no = n;
        bkg_mask = bkg;
}
//*************************************************************************

/*************************************************************************
        Function: makeBlock: this function divides a frame into blocks
        of block_size x block_size.
        input: frame array to be divided, its height and width, and the
        block size (square block sizes (e.g. 4x4 at level 2)are used for
        simplicity)
        output: array of block structure with number of blocks and the
        address (x,y) of the top-right corner of each block
*************************************************************************/
void GMEC::makeBlock(float a[FRAME_HEIGHT][FRAME_WIDTH], BLOCK *b,
                     int h, int w, int block_size)
{
        int bw, bh, count=0;
        register int i, j, m, n;

        bw = int(w/block_size); //width of block array
        bh = int(h/block_size); //height of block array

        for (i=0; i<bh; i++) {
               for (j=0; j<bw; j++) {
                      for (m=0; m<block_size; m++) {
                             for (n=0; n<block_size; n++) {
                                    b[i*bw+j].bvalue[m*block_size + n] =
                                           a[i*block_size+m][j*block_size+n];
                             }
                      }
                      b[i*bw+j].bx = j*block_size;
                      b[i*bw+j].by = i*block_size;
                      count++;
               }
        }
}

/*************************************************************************
        Function: buildSearchArea: this function builds search area for
        each block of a frame within 2-pixels around the block.
        input: frame array, its height and width, and the
        block size (square block sizes (e.g. 4x4 at level 2)are used for
        simplicity)
        output: array of search area structure with search size of each
        block, and the candidate motion vectors of each block in a
        search area.
*************************************************************************/
void GMEC::buildSearchArea (float pyramid[FRAME_HEIGHT][FRAME_WIDTH],
                            SEARCHAREA *SA1,MV *mv, int h,int w,
                            int block_size)
{
        register int i, j, m, n, x, y;
        int mstart,mend,nstart,nend,hb,wb;
        int count1=0, count2=0, scale;

        hb    = int(h/block_size);
        wb    = int(w/block_size);

        scale = 2;    //scales motion vectors to lower level pyramids

        for (i=0; i<hb; i++)
        {
               //define search area
               //the search area is a square block defined as
               //p + block size + p where p=2 pixels
               //so the search area extends from -2 to 2 horizontally
               //and -2 to 2 vertically
```

```
                if(((mv[i].mvy)*scale+i*block_size)<=0) //upper pixels
                        mstart = 0;
                else
                        mstart = -2;
                //lower pixels:
                if(((mv[i].mvy)*scale+i*block_size)>=((hb-1)*block_size))
                        mend = 0;
                else
                        mend = 2;
                for(j=0; j<wb; j++)
                {
                        if(((mv[j].mvx)*scale+j*block_size)<=0) //left most pixels
                                nstart = 0;
                        else
                                nstart = -2;
                        if(((mv[j].mvx)*scale+j*block_size)>=((wb-1)*block_size))
                                //rightmost pixels
                                nend = 0;
                        else
                                nend = 2;
                        for(m=mstart; m<=mend; m++)
                        {
                                for(n=nstart; n<=nend; n++)
                                {
                                        for(y=0; y<block_size; y++)
                                        {
                                                for(x=0; x<block_size; x++)
                                                {
        SA1[count2].bs[count1].bvalue[y*block_size+x] =
                                                pyramid[i*block_size+scale*(mv[i].mvy)+m+y][j*
                                                        block_size+scale*(mv[j].mvx)+n+x];
                                                }
                                        }
                                        SA1[count2].bs[count1].bx = scale*(mv[j].mvx)+n;
                                        SA1[count2].bs[count1].by = scale*(mv[i].mvy)+m;
                                        count1++;
                                }
                        }
                        SA1[count2].searchsize = count1;
                        count1 = 0;
                        count2++;
                }
        }
}
//*************************************************************************

/******************** block matching for level-2 ************************
        this function performs block matching for a current block for
        level-2. first, MADs within a search area are computed for each
        block. then three candidate blocks are selected for a block whose
        motion vectors are to be propagated to next level. the resulting
        motion vectors are assigned to a mv struct
*************************************************************************/

void GMEC::BMALevel2(BLOCK *b, SEARCHAREA *SA, MV *mv1,MV *mv2,MV *mv3,
                     int block_size)
{
        register int i, j, index;
        float *MAD=0, opt_mad;

        MAD = new float[MAX_SEARCH_AREA];

        //test for mem alloc for MAD
        if(!(MAD = new float[MAX_SEARCH_AREA])){
                cout<<"Memory allocation error for MAD array. "<<endl;
                exit(1);
        }


        for (i=0; i<NUM_BLOCKS; i++) {
```

```
            for (j=0; j<SA[i].searchsize; j++) {
                  //block match
                  MAD[j] = computeMAD(b[i],SA[i].bs[j], block_size*block_size);
            }

            //compute first optimal MAD
            index = MADopt(MAD, SA[i].searchsize, opt_mad);
            mv1[i].mvx = SA[i].bs[index].bx;
            mv1[i].mvy = SA[i].bs[index].by;
            //reset first optimal MAD
            MAD[index] = 255;   //max MAD

            //compute second optimal MAD
            index = MADopt(MAD, SA[i].searchsize, opt_mad);

            //save second candidate mv
            mv2[i].mvx = SA[i].bs[index].bx;
            mv2[i].mvy = SA[i].bs[index].by;

            //reset second optimal MAD
            MAD[index] = 255;   //max MAD

            //compute third optimal MAD
            index = MADopt(MAD, SA[i].searchsize, opt_mad);

            //save third candidate mv
            mv3[i].mvx = SA[i].bs[index].bx;
            mv3[i].mvy = SA[i].bs[index].by;
      }

      delete [] MAD;
}
//***************************************************************************

/**************** block matching for level-1 and level-0 *****************
      same as block matching for level-2, but MADs are computed for
      3 candidate blocks propagated from level-2 for level-1 and from
      level-1 for level-0. at level-0, the final optimal mvs are
      selected with the smallest MAD
*************************************************************************/
void GMEC::BMA(BLOCK *b, SEARCHAREA *SA1, SEARCHAREA *SA2,SEARCHAREA *SA3,
               MV *mv, MV *mv1, MV *mv2, MV *mv3, int block_size)
{
      register int  i, j1, j2, j3;
      int opt_index=0, index[3]={0};
      float madopt1=0, madopt2=0, madopt3=0, *MAD1=0, *MAD2=0, *MAD3=0;

      //MAD arrays for 3 mvs propagated
      MAD1 = new float[MAX_SEARCH_AREA];
      MAD2 = new float[MAX_SEARCH_AREA];
      MAD3 = new float[MAX_SEARCH_AREA];

      //test for mem allocation for MAD arrays
      if(!(MAD1||MAD2||MAD3)){
            cout<<"Memory allocation error for MAD arrays. "<<endl;
            exit(1);
      }

      //run through all blocks in a frame
      for (i=0; i<NUM_BLOCKS; i++) {
            for (j1=0; j1<SA1[i].searchsize; j1++) {
                  //block match for candidate mv-1
                  MAD1[j1] = computeMAD(b[i],SA1[i].bs[j1],
                                        block_size*block_size);
            }
            for (j2=0; j2<SA2[i].searchsize; j2++) {
                  //block match for candidate mv-2
                  MAD2[j2] = computeMAD(b[i],SA2[i].bs[j2],
                                        block_size*block_size);
            }
            for (j3=0; j3<SA3[i].searchsize; j3++) {
```

```
                        //block match for candidate mv-3
                        MAD3[j3] = computeMAD(b[i],SA3[i].bs[j3],
                                              block_size*block_size);
                }
                //compute optimal MADs
                index[0] = MADopt(MAD1, SA1[i].searchsize, madopt1);
                index[1] = MADopt(MAD2, SA2[i].searchsize, madopt2);
                index[2] = MADopt(MAD3, SA3[i].searchsize, madopt3);

                //save candidate mvs (works for level-1 only)
                mv1[i].mvx = SA1[i].bs[index[0]].bx;
                mv1[i].mvy = SA1[i].bs[index[0]].by;
                mv2[i].mvx = SA2[i].bs[index[1]].bx;
                mv2[i].mvy = SA2[i].bs[index[1]].by;
                mv3[i].mvx = SA3[i].bs[index[2]].bx;
                mv3[i].mvy = SA3[i].bs[index[2]].by;

                //compute smallest MAD, save mv with smallest mad
                if((madopt1<=madopt2) && (madopt1<=madopt3)){
                        mv[i].mvx = mv1[i].mvx;
                        mv[i].mvy = mv1[i].mvy;
                } else if(madopt2<=madopt3){
                        mv[i].mvx = mv2[i].mvx;
                        mv[i].mvy = mv2[i].mvy;
                }else{
                        mv[i].mvx = mv3[i].mvx;
                        mv[i].mvy = mv3[i].mvy;
                }
        }

        //free memory allocated
        delete [] MAD1;
        delete [] MAD2;
        delete [] MAD3;
}
//*************************************************************************

//****************** hierarchical motion estimation **********************
/*
        This function first builds three-level pyramids of the two frames
        previous and current it then performs hierarchical block matching.
        mv's at higher levels are propagated to next levels. for each level,
        it calls makeBlock() to divide current frame into blocks. it also calls
        buildSearchArea() to structure the previous frame into search areas
        corresponding to blocks of the current frame. block matching is then
        performed to find motion vectors at each level.
        the final mv's are then saved in a text file
*/
//*************************************************************************

void GMEC::mEstimate()
{
        register int i, j;

        //build 3-level pyramids for the two frames:

        //level-0 of pyramids corresponds to original images
        int          block_size0;
        int          h0, w0;

        h0 = orig_frame_height;
        w0 = orig_frame_width;
        block_size0 = BLOCK_SIZE;
        int h1, w1, block_size1;

        //level-1 of pyramids
        float  (*p11)[FRAME_WIDTH] = new float[FRAME_HEIGHT][FRAME_WIDTH];
        float  (*p12)[FRAME_WIDTH] = new float[FRAME_HEIGHT][FRAME_WIDTH];

        h1 = int(h0/2);
        w1 = int(w0/2);
```

```
block_size1 = int(block_size_0/2);

//construct level-1 mean frames of pyramids 1 and 2 from
//level 0 frames previous and current y-frames
meanPyramid(previous_y_frame, p11, h0, w0);
meanPyramid(current_y_frame, p12, h0, w0);

//level-2 of pyramids
float  (*p21)[FRAME_WIDTH] = new float[FRAME_HEIGHT][FRAME_WIDTH];
float  (*p22)[FRAME_WIDTH] = new float[FRAME_HEIGHT][FRAME_WIDTH];
int           h2, w2, block_size2;

h2          = int(h1/2);
w2          = int(w1/2);
block_size2 = int(block_size1/2);

//construct level-2 mean frames of pyramids 1 and 2 from
//level 1 frames p11 and p12
meanPyramid(p11, p21, h1, w1);
meanPyramid(p12, p22, h1, w1);

//hierarchical motion estimation

//allocate and initialize mem for motion vectors of blocks
MV *mv   = new MV[NUM_BLOCKS];
//3 mvs at level 2
MV *mv21 = new MV[NUM_BLOCKS];
MV *mv22 = new MV[NUM_BLOCKS];
MV *mv23 = new MV[NUM_BLOCKS];
//3 mvs at level 1
MV *mv11  = new MV[NUM_BLOCKS];
MV *mv12  = new MV[NUM_BLOCKS];
MV *mv13  = new MV[NUM_BLOCKS];
//3 mvs at level 0 + final mv
MV *mv01  = new MV[NUM_BLOCKS];
MV *mv02  = new MV[NUM_BLOCKS];
MV *mv03  = new MV[NUM_BLOCKS];

//Test for memory allocation errors for motion vectors
if(!(mv||mv21||mv22||mv23||mv11||mv12||mv13||
             mv01||mv02||mv03))
{
       cout<<"Memory allocation error for motion vectors";
              <<"in hierarchical motion estimation.";
              <<endl;
       exit(1);
}

//initialize motion vectors to zero
for(i=0; i<NUM_BLOCKS; i++){
       mv[i].mvx  = 0;            mv[i].mvy  = 0;

       mv21[i].mvx = 0;           mv21[i].mvy = 0;
       mv22[i].mvx = 0;           mv22[i].mvy = 0;
       mv23[i].mvx = 0;           mv23[i].mvy = 0;

       mv11[i].mvx = 0;           mv11[i].mvy = 0;
       mv12[i].mvx = 0;           mv12[i].mvy = 0;
       mv13[i].mvx = 0;           mv13[i].mvy = 0;

       mv01[i].mvx = 0;           mv01[i].mvy = 0;
       mv02[i].mvx = 0;           mv02[i].mvy = 0;
       mv03[i].mvx = 0;           mv03[i].mvy = 0;
}

//allocate mem for blocks
BLOCK *b2 = new BLOCK[NUM_BLOCKS];
BLOCK *b1 = new BLOCK[NUM_BLOCKS];
BLOCK *b0 = new BLOCK[NUM_BLOCKS];

//test for mem allocation errors for blocks
```

```
if(!(b2||b1||b0)){
	cout<<"Mem allocation error for blocks in hierarchical me. ";
	exit(1);
}

//initialize values of blocks to zero
for(i=0; i<NUM_BLOCKS; i++){
	b2[i].bvalue = new float[BLOCK_AREA];
	b1[i].bvalue = new float[BLOCK_AREA];
	b0[i].bvalue = new float[BLOCK_AREA];
}

//allocate mem for search areas
SEARCHAREA *SA2 = new SEARCHAREA[NUM_BLOCKS];
SEARCHAREA *SA11 = new SEARCHAREA[NUM_BLOCKS];
SEARCHAREA *SA12 = new SEARCHAREA[NUM_BLOCKS];
SEARCHAREA *SA13 = new SEARCHAREA[NUM_BLOCKS];
SEARCHAREA *SA01 = new SEARCHAREA[NUM_BLOCKS];
SEARCHAREA *SA02 = new SEARCHAREA[NUM_BLOCKS];
SEARCHAREA *SA03 = new SEARCHAREA[NUM_BLOCKS];

//Test for mem allocation errors for search areas
if(!(SA2||SA11||SA12||SA13||SA01||SA02||SA03)){
	cout<<"Mem allocation error for search areas ";
		<<"in hierarchical me.";
		<<endl;
	exit(1);
}

//initialize search areas
for(i=0; i<NUM_BLOCKS; i++){

	//MAX_SEARCH_AREA = 25 for 16x16 block
	SA2[i].bs = new BLOCK[MAX_SEARCH_AREA];

	SA11[i].bs = new BLOCK[MAX_SEARCH_AREA];
	SA12[i].bs = new BLOCK[MAX_SEARCH_AREA];
	SA13[i].bs = new BLOCK[MAX_SEARCH_AREA];

	SA01[i].bs = new BLOCK[MAX_SEARCH_AREA];
	SA02[i].bs = new BLOCK[MAX_SEARCH_AREA];
	SA03[i].bs = new BLOCK[MAX_SEARCH_AREA];

	for(j=0; j<MAX_SEARCH_AREA; j++){
		SA2[i].bs[j].bvalue = new float[BLOCK_AREA];

		SA11[i].bs[j].bvalue = new float[BLOCK_AREA];
		SA12[i].bs[j].bvalue = new float[BLOCK_AREA];
		SA13[i].bs[j].bvalue = new float[BLOCK_AREA];

		SA01[i].bs[j].bvalue = new float[BLOCK_AREA];
		SA02[i].bs[j].bvalue = new float[BLOCK_AREA];
		SA03[i].bs[j].bvalue = new float[BLOCK_AREA];
	}
}

//block matching and hierarchical motion vector estimation

//level-2
makeBlock(p22, b2, h2, w2, block_size2);
buildSearchArea(p21, SA2, mv, h2, w2, block_size2);

	//block match
BMALevel2(b2, SA2, mv21,mv22,mv23, block_size2);

//level-1
makeBlock(p12, b1, h1, w1, block_size1);
buildSearchArea(p11, SA11, mv21, h1, w1, block_size1);
buildSearchArea(p11, SA12, mv22, h1, w1, block_size1);
buildSearchArea(p11, SA13, mv23, h1, w1, block_size1);
```

```
      //block match
BMA(b1, SA11, SA12, SA13, mv, mv11, mv12, mv13, block_size1);

//level-0
float (*g1)[FRAME_WIDTH] = new float[FRAME_HEIGHT][FRAME_WIDTH];
float (*g2)[FRAME_WIDTH] = new float[FRAME_HEIGHT][FRAME_WIDTH];

normalizeFrames(previous_y_frame,current_y_frame,g1,g2,
                FRAME_HEIGHT, FRAME_WIDTH);
makeBlock(g2, b0, h0, w0, block_size0);
buildSearchArea(g1, SA01, mv11, h0, w0, block_size0);
buildSearchArea(g1, SA02, mv12, h0, w0, block_size0);
buildSearchArea(g1, SA03, mv13, h0, w0, block_size0);

      //block match
BMA(b0, SA01, SA02, SA03, mv, mv01, mv02, mv03, block_size0);

for(i=0; i<NUM_BLOCKS; i++){
      orig_block[i].bx = b0[i].bx;
      orig_block[i].by = b0[i].by;
      mv_final[i].mvx = mv[i].mvx;
      mv_final[i].mvy = mv[i].mvy;
}

//save final motion vectors - mv_final
saveMotionVectors();


//free memory allocated
delete [] p11;             delete [] p12;
delete [] p21;             delete [] p22;

delete [] mv;              delete [] mv21;
delete [] mv22;            delete [] mv23;
delete [] mv11;            delete [] mv12;
delete [] mv13;

delete [] mv01;            delete [] mv02;
delete [] mv03;

for(i=0; i<NUM_BLOCKS; i++){
      delete [] b2[i].bvalue;
      delete [] b1[i].bvalue;
      delete [] b0[i].bvalue;
}
delete [] b2;
delete [] b1;
delete [] b0;

for(i=0; i<NUM_BLOCKS; i++){
      for(j=0; j<MAX_SEARCH_AREA; j++){
            delete [] SA2[i].bs[j].bvalue;

            delete [] SA11[i].bs[j].bvalue;
            delete [] SA12[i].bs[j].bvalue;
            delete [] SA13[i].bs[j].bvalue;

            delete [] SA01[i].bs[j].bvalue;
            delete [] SA02[i].bs[j].bvalue;
            delete [] SA03[i].bs[j].bvalue;
      }
      delete [] SA2[i].bs;

      delete [] SA11[i].bs;
      delete [] SA12[i].bs;
      delete [] SA13[i].bs;

      delete [] SA01[i].bs;
      delete [] SA02[i].bs;
      delete [] SA03[i].bs;
}
```

```
	delete [] SA2;

	delete [] SA11;
	delete [] SA12;
	delete [] SA13;

	delete [] SA01;
	delete [] SA02;
	delete [] SA03;

	delete [] g1;
	delete [] g2;
}
//*************************************************************************

/********************** save final motion vectors ***********************
	this function saves the final block motion vectors.
	x and y components of final mv's are saved to files "mvx.txt"
	and "mvy.txt".
*************************************************************************/
void GMEC::saveMotionVectors()
{
	register int i, j;
	//open file for output
	ofstream outmvx("mvx.txt", ios::app|ios::binary);
	ofstream outmvy("mvy.txt", ios::app|ios::binary);

	//save motion vector arrays
	outmvx<<"\nMotion vectors between frames: "<<frame_no;
	outmvx<<" and "<<frame_no+1;
	outmvx<<"\n";
	outmvy<<"\nMotion vectors between frames: "<<frame_no;
	outmvy<<" and "<<frame_no+1;
	outmvy<<"\n";
	for (i=0; i<BLOCK_HEIGHT; i++){
		for(j=0; j<BLOCK_WIDTH; j++){
			outmvx<<mv_final[i*BLOCK_WIDTH+j].mvx<<' ';
			outmvy<<mv_final[i*BLOCK_WIDTH+j].mvy<<' ';
		}
		outmvx<<"\n";
		outmvy<<"\n";
	}
}
//*************************************************************************

//********************** affine parameter estimation ***********************
/*
	this function performs the 6-parameter affine estimation from
	block motion vectors. before parameter estimation, motion vectors
	which does not belong to background are rejected. also, mv's are
	are smoothed according to their neighborhood
*/
//*************************************************************************
void GMEC::affineEstmate()
{
	int block_size = BLOCK_SIZE;

	//matrix of the 6 systems of equations
	double s[6][6]={0}, b[6]={0};

	register int i, j;
	register int m, n; //for calcualting mean of 3x3 groups

	//threshold=1/4-pixel threshold to reject mvs>> or<< mean of 3x3 groups
	const float threshold = 0.25;
	double x=0, y=0, vx=0, vy=0, tmpx=0, tmpy=0, tmpsum=0,
		sumx=0, sumxsq=0, sumy=0, sumysq=0, sumxy=0,sum1=0,
		sumxvx=0, sumyvx=0, sumvx=0, sumxvy=0, sumyvy=0, sumvy=0;
	float  x_mean=0, y_mean=0;
	float  temp_sum=0;
	//temp arrays to hold mvs and x and y coords ofmvs
```

```
float (*mvx_array)[FRAME_WIDTH]= new float[FRAME_HEIGHT][FRAME_WIDTH];
float (*mvy_array)[FRAME_WIDTH]= new float[FRAME_HEIGHT][FRAME_WIDTH];
float (*median_mvx)[FRAME_WIDTH]= new float[FRAME_HEIGHT][FRAME_WIDTH];
float (*median_mvy)[FRAME_WIDTH]= new float[FRAME_HEIGHT][FRAME_WIDTH];
int      (*x_array)[FRAME_WIDTH] = new int[FRAME_HEIGHT][FRAME_WIDTH];
int   (*y_array)[FRAME_WIDTH] = new int[FRAME_HEIGHT][FRAME_WIDTH];

//phase 1: outlier rejection, field smoothing

//first convert motion vector structs to arrays
//also convert x and y coords to arrays
for (i=0; i<BLOCK_HEIGHT; i++){
       for (j=0; j<BLOCK_WIDTH; j++){
              for(m=0; m<BLOCK_SIZE; m++){
                     for(n=0; n<BLOCK_SIZE; n++){
                            mvx_array[i*block_size+m][j*block_size+n] =
                                   float(mv_final[i*BLOCK_WIDTH+j].mvx);
                            mvy_array[i*block_size+m][j*block_size+n] =
                                   float(mv_final[i*BLOCK_WIDTH+j].mvy);

                            x_array[i*block_size+m][j*block_size+n] =
                                   j*block_size+n;
                            y_array[i*block_size+m][j*block_size+n] =
                                   i*block_size+m;
                     }
              }
       }
}
       //smooth mvs
double ux, uy, fix, fiy, smoothed_sumx=0, smoothed_sumy=0;
int h_, temp_abs, div=0;
for (i=10; i<FRAME_HEIGHT-10; i++) {
       for (j=10; j<FRAME_WIDTH-10; j++){

              //calculate mean for 3x3 group of pixels
              for(m=-1; m<=1; m++){
                     for(n=-1; n<=1; n++){
                            ux = mvx_array[i][j];
                            uy = mvy_array[i][j];
                            fix = fabs(mvx_array[i+m][j+n]-ux);
                            fiy = fabs(mvy_array[i+m][j+n]-uy);
                            if(fix<=0.125 && fiy<=0.125){
                                   fix = 1;
                                   fiy = 1;
                            }else{
                                   fix = 0;
                                   fiy = 0;
                            }
                            temp_abs = abs(m)+abs(n);
                            if(temp_abs==0) h_=4;
                            else if(temp_abs==1) h_=2;
                            else h_=1;

                            smoothed_sumx += fix*h_+ux;
                            smoothed_sumy += fiy*h_+uy;
                            div += int(fix*h_);

                     }
              }
              median_mvx[i][j] = float(smoothed_sumx/div) + j;
              median_mvy[i][j] = float(smoothed_sumy/div) + i;
       }
}

//reject mvs that do not belong to background
//reject edge blocks
//compute sums
for (i=10; i<FRAME_HEIGHT-10; i++) {
       for (j=10; j<FRAME_WIDTH-10; j++){
              if(bkg_mask[i][j]>=1){
```

```
                        //compute sums
                        vx = median_mvx[i][j];
                        vy = median_mvy[i][j];
                        x  = x_array[i][j];
                        y  = y_array[i][j];

                        sumxsq += x*x;
                        sumxy  += x*y;
                        sumx   += x;
                        sumysq += y*y;
                        sumy   += y;
                        sum1   += 1;

                        sumxvx += x*vx;
                        sumyvx += y*vx;
                        sumvx  +=  vx;
                        sumxvy += x*vy;
                        sumyvy += y*vy;
                        sumvy  +=  vy;
                    }
                }
        }

        //phase-2: matrix for system of equations

              //populate coefficient arrays s[][] and b[]
        s[0][0] = sumxsq; s[0][1] = sumxy;  s[0][2] = sumx;
        s[0][3] = 0;      s[0][4] = 0;      s[0][5] = 0;

        s[1][0] = sumxy;  s[1][1] = sumysq; s[1][2] = sumy;
        s[1][3] = 0;      s[1][4] = 0;      s[1][5] = 0;

        s[2][0] = sumx;   s[2][1] = sumy;   s[2][2] = sum1;
        s[2][3] = 0;      s[2][4] = 0;      s[2][5] = 0;

        s[3][0] = 0;      s[3][1] = 0;      s[3][2] = 0;
        s[3][3] = sumxsq; s[3][4] = sumxy;  s[3][5] = sumx;

        s[4][0] = 0;      s[4][1] = 0;      s[4][2] = 0;
        s[4][3] = sumxy;  s[4][4] = sumysq; s[4][5] = sumy;

        s[5][0] = 0;      s[5][1] = 0;      s[5][2] = 0;
        s[5][3] = sumx;   s[5][4] = sumy;   s[5][5] = sum1;

        b[0] = sumxvx;        b[1] = sumyvx;    b[2] = sumvx;
        b[3] = sumxvy;        b[4] = sumyvy;    b[5] = sumvy;

        //phase-3: parameter estimation
        //compute param vector using gauss elimination
        gauss(s,b,A);

        //free mem
        delete [] mvx_array;
        delete [] mvy_array;
        delete [] median_mvx;
        delete [] median_mvy;
        delete [] x_array;
        delete [] y_array;
}
//*************************************************************************

//*************************** frame warping ******************************
/*
        this function performs global motion compensation according to the
        estimated affine parameters. it uses bicubic interpolation for
        non-integer pixel locations
*/
//*************************************************************************
void GMEC::warpFrame()
{
        double x_=0, y_=0;
```

```
	register int x, y;
	int x1, x2, y1, y2;
	float q11, q12, q21, q22, value1, value2,

	//do motion estimation
	mEstimate();

	//do affine param estimation
	affineEstmate();

	//for each pixel of warped frame, compute coords of corresponding
	//pixel in previous frame
	for (y=0; y<FRAME_HEIGHT; y++){
		for (x=0; x<FRAME_WIDTH; x++){
			x_ = A[0]*x + A[1]*y + A[2];
			y_ = A[3]*x + A[4]*y + A[5];

			x1 = int(x_);
			y1 = int(y_);

			//ignore pixels whose new coords are out of boarder
			if(x1>=0 && y1>=0 && x1<=FRAME_WIDTH-1 && y1<=FRAME_HEIGHT-1)
			{
				//bilinear interpolation

				x2 = x1+1;
				y2 = y1+1;

				//y-frame
				q11 = previous_y_frame[y1][x1];
				q12 = previous_y_frame[y1][x2];
				q21 = previous_y_frame[y2][x1];
				q22 = previous_y_frame[y2][x2];
				value1 = (float)(x2-x1)*(y2-y1);
				value2 = q11*(x2-x)*(y2-y)/value1 +
							q21*(x-x1)*(y2-y)/value1 +
							q12*(x2-x)*(y-y1)/value1 +
							q22*(x-x1)*(y-y1)/value1 ;

				warped_y_frame[y][x] = value2_y;
			}
			else{
				warped_y_frame[y][x] = 0;
			}
		}
	}

}
//*************************************************************************

//****************** function to obtain warped frames ********************
//this function is used to access warped frames, which are the
//results of global motion estimation and compensation

void GMEC::getWarped(float y[][FRAME_WIDTH])
{
	int i, j;

	warpFrame();

	for(i=0; i<FRAME_HEIGHT; i++){
		for(j=0; j<FRAME_WIDTH; j++){
			y[i][j] = warped_y_frame[i][j];
		}
	}
}
//*************************************************************************

//************* destructor implementation: free allocated mem *************
GMEC::~GMEC()
{
```

```
        register int i;

        delete [] previous_y_frame;
        delete [] current_y_frame;
        delete [] warped_y_frame;
        for(i=0; i<NUM_BLOCKS; i++){
              delete [] orig_block[i].bvalue;
        }
        delete [] mv_final;
        delete [] orig_block;
}
//*************************************************************************

//****************** end of GMEC class implementation *********************
```

B-2 CD Class

```
//*************************************************************************
/*
        Header file for statistical change detection class
        Header file for Hierarchical Mean Pyramid based
        Global motion estimation and compensation
        Written by: Eneyachew Tamir
        MSc project
        Addis Ababa University
        Aug 2007

  */
//*************************************************************************

#ifndef _SVO_UTIL_H_
#define _SVO_UTIL_H_

#include "svo_util.h"

class CD
{
public:
        CD();
        ~CD();

        //methods
        void initCD(float y_plane_prev[][FRAME_WIDTH],
                    float u_plane_prev[][FRAME_WIDTH],
                    float v_plane_prev[][FRAME_WIDTH],
                    unsigned char *current_frame,
                    int (*prev_bkg)[FRAME_WIDTH], int h, int w);
        void getBkgmask(int [][FRAME_WIDTH]);
        void getObjmask(int bm[][FRAME_WIDTH],int [][FRAME_WIDTH]);

private:
        //methods
        void convertFrame(unsigned char *frame);
        void normalizeFrames(float g[][FRAME_WIDTH]);
        float noiseSd();
        void frameDifference();
        void produceMask();

        //attributes
        unsigned char *previous_frame;
        unsigned char *current_frame;
        float (*y_prev)[FRAME_WIDTH];
        float (*y_curr)[FRAME_WIDTH];
        float (*di)[FRAME_WIDTH];
        int (*bkg_mask)[FRAME_WIDTH];
        int (*obj_mask)[FRAME_WIDTH];
        int (*prev_bkg_mask)[FRAME_WIDTH];
```

```
	int frame_h;
	int frame_w;
};
#endif	//_SVO_UTIL_H_

//********************** end of CD class header file ***********************

//*****************************************************************************
 /*
		Implementation for statistical change detection class
		Written by: Eneyachew Tamir
		MSc project
		Addis Ababa University
		Aug 2007
  */
//*****************************************************************************

#include <math.h>
#include "svo_CD.h"
#include <fstream.h>
#include <iostream.h>
#include <algorithm>
#include <functional>
#include <iterator>

using namespace std;

//********************** constructor implementation ************************
CD::CD()
{
	//allocate mem for arrays
	previous_frame = new unsigned char[YUV_SIZE];
	current_frame = new unsigned char[YUV_SIZE];

	y_prev = new float[FRAME_HEIGHT][FRAME_WIDTH];
	y_curr = new float[FRAME_HEIGHT][FRAME_WIDTH];

	bkg_mask = new int[FRAME_HEIGHT][FRAME_WIDTH];
	obj_mask = new int[FRAME_HEIGHT][FRAME_WIDTH];
	prev_bkg_mask = new int[FRAME_HEIGHT][FRAME_WIDTH];

	di = new float[FRAME_HEIGHT][FRAME_WIDTH];

	//initialize arrays
	for(int i=0; i<FRAME_HEIGHT; i++){
		for(int j=0; j<FRAME_WIDTH; j++){
			y_prev[i][j] = 0;
			y_curr[i][j] = 0;
			bkg_mask[i][j] = 0;
			obj_mask[i][j] = 0;
			prev_bkg_mask[i][j] = 1;
			di[i][j] = 0;
		}
	}
}
//*****************************************************************************

//********************** initialize change detection ***********************
//function to initialize change detection
void CD::initCD(float (*y_plane_prev)[FRAME_WIDTH],
			unsigned char *current_frame,
			int (*prev_bkg)[FRAME_WIDTH], int h, int w)
{
	register int i, j;

	frame_h = h;
	frame_w = w;

	for(i=0; i<FRAME_HEIGHT; i++){
		for(j=0; j<FRAME_WIDTH; j++){
			y_prev[i][j] = y_plane_prev[i][j];
```

```
                        prev_bkg_mask[i][j] = prev_bkg[i][j];
                }
        }
}
//*************************************************************************

//************************* frame normalization ************************
/*
        this function takes two frames f1 and f2, computes mean and variance
        of the two, normalizes the first frame with respect to the second
        based on second order statistics inside a 3x3 window the normalized
        previous is assigned to g
*/
//*************************************************************************
void CD::normalizeFrames(float (*g)[FRAME_WIDTH])
{
        register int x, y, m, n;
        int mstart=-2, mend=2, nstart=-2, nend=2;
        float mean1=0, mean2=0, var1=0, var2=0,tmpval;

        //go through all pixels, ignoring edge pixels
        for (y=0;y<FRAME_HEIGHT; y++){
                for (x=0; x<FRAME_WIDTH; x++){

                        //ignore edge pixels
                        if((y>=2)&&(y<FRAME_HEIGHT-2)&&(x>=2)&&(x<FRAME_WIDTH-2))
                        {
                                //compute mean for the 3x3 window
                                for(m=-1; m<=1; m++){
                                        for(n=-1; n<=1; n++){
                                                mean1 += y_prev[y+m][x+n];
                                                mean2 += y_curr[y+m][x+n];
                                        }
                                }
                                mean1 /= 9;
                                mean2 /= 9;

                                //compute variance for the 3x3 window
                                for(m=-1; m<=1; m++){
                                        for(n=-1; n<=1; n++){
                                                var1 += (y_prev[y+m][x+n] -
                                                 mean1)*(y_prev[y+m][x+n] - mean1);
                                                var2 += (y_curr[y+m][x+n] -
                                                 mean2)*(y_curr[y+m][x+n] - mean2);
                                        }
                                }
                                var1 /= 9;
                                var2 /= 9;

                                //compute normalized frame
                                tmpval = (var1/var2)*(y_prev[y][x]-mean2) + mean2;
                                if(tmpval>256)
                                        g[y][x] = 256;
                                else if (tmpval<0)
                                        g[y][x] = 0;
                                else
                                        g[y][x] = tmpval;
                                //reset variables
                                var1  = var2  = 0;
                                mean1 = mean2 = 0;
                        }
                        else
                                g[y][x] = y_prev[y][x];
                }
        }
}
//*************************************************************************

//******** function to compute background noise from prev bkg mask ********
float CD::noiseSd()
{
```

```
	register int i,j;
	int count=0;
	double mean=0, var=0, sd=0;
	float temp[Y_SIZE] = {0};

	//filter background differece
	for(i=0; i<FRAME_HEIGHT; i++){
		for(j=0; j<FRAME_WIDTH; j++){
			if(prev_bkg_mask[i][j]==1){
				temp[count] = di[i][j];
				mean += di[i][j];
				++count;
			}
		}
	}
	//compute mean of background difference
	mean /= count;
	//compute variance

	for(i=0; i<count; i++){
		var += (temp[i] - mean)*(temp[i] - mean);
	}
	var /= count;
	sd = sqrt(var);
	return sd;
}

//******************** function for frame differencing *******************
/*
	this function computes absolute differe between two frames
	first, it applies normalization by using normalizeFrame() method
	then it subtracts previous frame from current frame, and applies
	absolute value to the difference. the result is assigned to the
	variable di which is global to the class
*/
//*************************************************************************

void CD::frameDifference()
{
	register int x, y;

	//arrays for frame normalization
	float (*y1)[FRAME_WIDTH] = new float[FRAME_HEIGHT][FRAME_WIDTH];

	//arrays for frame difference
	float (*dy)[FRAME_WIDTH] = new float[FRAME_HEIGHT][FRAME_WIDTH];

	for (y=0; y<FRAME_HEIGHT; y++){
		for (x=0; x<FRAME_WIDTH; x++){
			y1[y][x]=0;
			dy[y][x]=0;
		}
	}

	//normalize previous frame
	normalizeFrames(y1);

	//run through all pixels in the frames
	for (y=0; y<FRAME_HEIGHT; y++){
		for (x=0; x<FRAME_WIDTH; x++){
			di[y][x] = int(fabs(y_curr[y][x] - y1[y][x]));
		}
	}
	//free mem
	delete [] y1;
	delete [] dy;
}
//*************************************************************************

//*************************** classification ***************************
/*
```

```
	This function implements the statistical based classification of
	difference pixel according to the statistics of difference pixel
	inside a (e.g)5x5 window. to avoid misregistration errors, first
	gradient images are computed, then if the gradient of a pixel in both frames
	is high, the difference is avoided (nothing added to local sum)
	to avoid misregistration error. a threshold of 0.2 is used for gradient
	images.
	noise of background is computed, using previous background frame, and
	the local sum is then computed inside 5x5 window, and threshold is
	applied to the sum. if the local sum> the threshold t_alpha (from
	chi-square table) for a significant level alpha, the foreground is set,
	otherwise background is set to 1.
	some levels of significance alpha in the range 10e-2 to 10e-6 were
	tested
*/
//*********************************************************************
void CD::produceMask()
{
	register int  x, y, m, n;
	const double t_alpha=52.6;//=27.8 for 3x3, =52.6 for 5x5, =85.35for7x7
	//for significant level of 10e-3
	float		noise_sd=0, local_sum=0, delta_sq=0;
	const double grad_th=0.2;  //gradient threshold
	const int win=2;

	//apply frame differencing
	frameDifference();

	//compute noise sd for background part
	noise_sd = noiseSd();

	//for each difference pixel, compute local sum of overlapped windows
	//apply thresholding

	for (y=0; y<frame_h; y++){
		for (x=0; x<frame_w; x++){

			//reject edge pixels by setting to zero
			if((y>=8) && (x>=8) && (y<=frame_h-8) && (x<=frame_w-8)){
				//compute mean for the 5x5 window
				for(m=-win; m<=win; m++){
					for(n=-win; n<=win; n++){
						//compute local sum inside the window
		local_sum += (di[y+m][x+n]/noise_sd)* (di[y+m][x+n]/noise_sd);
						}
					}

				//apply thresholding
				//if local sum inside the window>threshold t_alpha,
				//objectmask is set to 1 otherwise bkg mask is set to 1
				if(local_sum>t_alpha){
					obj_mask[y][x] = 1;
					bkg_mask[y][x] = 0;
				}
				else{
					obj_mask[y][x] = 0;
					bkg_mask[y][x] = 1;
				}
			}
			else{
				obj_mask[y][x] = 0;
				bkg_mask[y][x] = 1;
			}
			local_sum = 0;
		}
	}
}
//*********************************************************************


//************ functions to obtain background and object masks ************
```

```
void CD::getObjmask(int bm[][FRAME_WIDTH],int om[][FRAME_WIDTH])
{
	register int x, y;

	//assign bkg mask
	for(y=0; y<FRAME_HEIGHT; y++){
		for(x=0; x<FRAME_WIDTH; x++){
			prev_bkg_mask[y][x] = bm[y][x];
		}
	}

	produceMask();

	for (y=0; y<frame_h; y++){
		for (x=0; x<frame_w; x++){
			om[y][x] = obj_mask[y][x];
		}
	}
}

void CD::getBkgmask(int bm[][FRAME_WIDTH])
{
	register int x, y;

	produceMask();
	for (y=0; y<frame_h; y++){
		for (x=0; x<frame_w; x++){
			bm[y][x] = bkg_mask[y][x];
		}
	}
}
//*************************************************************************

//****************************** Destructor ******************************
CD::~CD()
{
	delete [] previous_frame;
	delete [] current_frame;
	delete [] y_prev;
	delete [] u_prev;
	delete [] v_prev;

	delete [] y_curr;
	delete [] u_curr;
	delete [] v_curr;
	delete [] bkg_mask;
	delete [] obj_mask;
	delete [] prev_bkg_mask;
}
//*************************************************************************

//******************** end of CD class implementation *********************


//*************************************************************************
 /*
		Header file for post-processing class
		Written by: Eneyachew Tamir
		MSc project
		Addis Ababa University
		Aug 2007
  */
//*************************************************************************

#ifndef _SVO_PP_H_
#include "svo_util.h"

class PostP
{
public:
	PostP(){};
```

```
        ~PostP(){};

        void openP(int A[][FRAME_WIDTH], int C[][FRAME_WIDTH]);
        void closeP(int A[][FRAME_WIDTH], int C[][FRAME_WIDTH]);
        void combineMasks(int frame[][FRAME_WIDTH],int prev_mask[][FRAME_WIDTH],
                          int curr_mask[][FRAME_WIDTH],
                          int final_mask[][FRAME_WIDTH]);
        void superImpose(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
                         int [][FRAME_WIDTH]);
        void medianFilter2(int frame[][FRAME_WIDTH],float filtered[][FRAME_WIDTH],
                           int h, int w);
        void rightRaster(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
                         int result[][FRAME_WIDTH]);
        void leftRaster(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
                        int result[][FRAME_WIDTH]);
        void downRaster(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
                        int result[][FRAME_WIDTH]);
        void upRaster(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
                      int result[][FRAME_WIDTH]);

//private:
        void dilate(int A[][FRAME_WIDTH], int C[][FRAME_WIDTH]);
        void erode(int A[][FRAME_WIDTH], int C[][FRAME_WIDTH]);
};
#endif //_SVO_PP_H_

//********************** end of PP class header file **********************
```

B-3 PP Class

```
//*****************************************************************************
 /*
             Implementation file for post-processing PP class
             Written by: Eneyachew Tamir
             MSc project
             Addis Ababa University
             Aug 2007
  */
//*****************************************************************************

#include <iostream.h>
#include <math.h>
#include "svo_PP.h"
#include "svo_util.h"

//****************************** dilation *******************************
//this function takes the binary image to be dilated and performs
//dilation 5x5 structuring elements is used
void PostP::dilate(int A[][FRAME_WIDTH], int C[][FRAME_WIDTH])
{
        int N, M, K, L, n, m, k, l, dk, dl;

        //structuring element for dilation
               int B[5][5] = { {0,0,0,0,0},
                               {0,1,1,1,0},
                               {0,1,1,1,0},
                               {0,1,1,1,0},
                               {0,0,0,0,0} };

        N = FRAME_HEIGHT;          //binary image height
        M = FRAME_WIDTH;           //binary image width
        K = L = 2;
        dk = 2*K + 1;              //size of structuring element
        dl = 2*L + 1;

        //ignore pixels near edge:
        for(n=K; n<N-K; n++){
               for(m=L; m<M-L; m++){
```

```
                        C[n][m] = 0;
                        for(k=0; k<dk; k++){
                                for(l=0; l<dl; l++){
                                        C[n][m] |= ((B[k][l]) & (A[n+k-K][m+l-L]));
                                }
                        }
                }
        }
}
//*************************************************************************

//****************************** dilation ********************************
//this function takes the binary image to be dilated and performs
//dilation 5x5 and 3x3 structuring elements are used
void PostP::erode(int A[FRAME_HEIGHT][FRAME_WIDTH],
                                int C[FRAME_HEIGHT][FRAME_WIDTH])
{
        int N, M, K, L, n, m, k, l, dk, dl;

        //structuring element for erosion
                int B[5][5] = { {1,1,0,1,1},
                                {1,0,0,0,1},
                                {0,0,0,0,0},
                                {1,0,0,0,1},
                                {1,1,0,1,1} };

        N = FRAME_HEIGHT;           //binary image height
        M = FRAME_WIDTH;            //binary image width
        K = L = 2;
        dk = 2*K + 1;
        dl = 2*L + 1;

        //ignore pixels near edge:
        for(n=K; n<N-K; n++){

                for(m=L; m<M-L; m++){
                        C[n][m] = 1;
                        for(k=0; k<dk; k++){
                                for(l=0; l<dl; l++){
                                        C[n][m] &= ((B[k][l]) | (A[n+k-K][m+l-L]));
                                }
                        }
                }
        }
}
//*************************************************************************

//****************************** opening *********************************
void PostP::openP(int A[][FRAME_WIDTH], int C[][FRAME_WIDTH])
{
        int temp[FRAME_HEIGHT][FRAME_WIDTH] = {0};

        //apply erosion
        erode(A, temp);

        //apply dilation
        dilate(temp, C);

        for(int i=0; i<FRAME_HEIGHT; i++){
                for(int j=0; j<FRAME_WIDTH; j++){
                        if(C[i][j]>=1)
                                C[i][j]+=250;
                }

        }
}
//*************************************************************************

//****************************** closing *********************************
void PostP::closeP(int A[][FRAME_WIDTH], int C[][FRAME_WIDTH])
{
```

```
	int temp[FRAME_HEIGHT][FRAME_WIDTH] = {0};

	//apply dilation
	dilate(A, temp);
	//apply erosion
	erode(temp, C);

	for(int i=0; i<FRAME_HEIGHT; i++){
		for(int j=0; j<FRAME_WIDTH; j++){
			if(C[i][j]>=1)
				C[i][j]+=250;
		}
	}
}
//*************************************************************************

//********************** functions for filling-in ************************
//this function performs filling-in from left to right
void PostP::rightRaster(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
				int result[][FRAME_WIDTH])
{
	register int i, j;
	double grad=0;

	for(i=2; i<FRAME_HEIGHT-2; i++){
		for(j=2; j<FRAME_WIDTH-2; j++){
			grad = fabs(double(frame[i][j] - frame[i][j+1]))/256;

			if((mask[i][j]==0) && (mask[i][j-1]==0)){
				result[i][j]= 0;
				mask[i][j]= 0;
			}

			if(mask[i][j]>=1) {
				if((mask[i][j-1]==0) && (grad<0.04)){
					result[i][j]= 0;
					mask[i][j]= 0;
				}
				else{
					result[i][j]= 1;
					mask[i][j]= 1;
				}
			}

			if((mask[i][j]==0) && (mask[i][j-1]>=1)){
				if(grad<0.04){
					result[i][j]= 1;
					mask[i][j]= 1;
				}
				else{
					result[i][j]= 0;
					mask[i][j]= 0;
				}
			}
		}
	}
}

//this function performs filling-in from right to left
void PostP::leftRaster(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
					int result[][FRAME_WIDTH])
{
	register int i, j;
	double grad=0;

	for(i=FRAME_HEIGHT-2; i>2; i--){
		for(j=FRAME_WIDTH-2; j>2; j--){
			grad = fabs(double(frame[i][j] - frame[i][j-1]))/256;

			if((mask[i][j]==0) && (mask[i][j+1]==0)){
				mask[i][j]= 0;
```

```
                        result[i][j]= 0;
                }

                if(mask[i][j]>=1) {
                        if((mask[i][j+1]==0) && (grad<0.04)){
                                mask[i][j]= 0;
                                result[i][j]= 0;
                        }else{
                                result[i][j]= 1;
                                mask[i][j]= 1;
                        }
                }

                if((mask[i][j]==0) && (mask[i][j+1]>=1)){
                        if(grad<0.04){
                                result[i][j]= 1;
                                mask[i][j]= 1;
                        }
                        else{
                                result[i][j]= 0;
                                mask[i][j]= 0;
                        }
                }
        }
    }
}

//this function performs filling-in from top to bottom
void PostP::downRaster(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
                        int result[][FRAME_WIDTH])
{
    register int i, j;
    double grad=0;

    for(i=2; i<FRAME_HEIGHT-2; i++){
        for(j=2; j<FRAME_WIDTH-2; j++){
                grad = fabs(double(frame[i][j] - frame[i+1][j]))/256;

                if((mask[i][j]==0) && (mask[i-1][j]==0)){
                        result[i][j]= 0;
                        mask[i][j]= 0;
                }

                if(mask[i][j]>=1) {
                        if((mask[i-1][j]==0) && (grad<0.04)){
                                result[i][j]= 0;
                                mask[i][j]= 0;
                        }
                        else{
                                result[i][j]= 1;
                                mask[i][j]= 1;
                        }
                }

                if((mask[i][j]==0) && (mask[i-1][j]>=1)){
                        if(grad<0.04){
                                result[i][j]= 1;
                                mask[i][j]= 1;
                        }
                        else{
                                result[i][j]= 0;
                                mask[i][j]= 0;
                        }
                }


        }
    }
}

//this function performs filling-in from bottom to top
```

```
void PostP::upRaster(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
                     int result[][FRAME_WIDTH])
{
       register int i, j;
       double grad=0;

       for(i=FRAME_HEIGHT-2; i>2; i--){
              for(j=FRAME_WIDTH-2; j>2; j--){
                     grad = fabs(double(frame[i][j] - frame[i-1][j]))/256;

                     if((mask[i][j]==0) && (mask[i+1][j]==0)){
                            result[i][j]= 0;
                            mask[i][j]= 0;
                     }

                     if(mask[i][j]>=1) {
                            if((mask[i+1][j]==0) && (grad<0.04)){
                                   result[i][j]= 0;
                                   mask[i][j]= 0;
                            }
                            else{
                                   result[i][j]= 1;
                                   mask[i][j]= 1;
                            }
                     }

                     if((mask[i][j]==0) && (mask[i+1][j]>=1)){
                            if(grad<0.04){
                                   result[i][j]= 1;
                                   mask[i][j]= 1;
                            }
                            else{
                                   result[i][j]= 0;
                                   mask[i][j]= 0;
                            }
                     }
              }
       }
}

//************************* combining CD masks **************************
//this function takes two change detection masks, prev_mask and
//curr_mask, combines the two by using logical AND operation, performs forward,
//backward, upward and downward filling-in. the final result is assigned to the
//variable final_mask

void PostP::combineMasks(int frame[][FRAME_WIDTH],int right_mask[][FRAME_WIDTH],
                         int left_mask[][FRAME_WIDTH],
                         int final_mask[][FRAME_WIDTH])
{
       register int i, j, x, y;
       int    (*raster1)[FRAME_WIDTH],(*raster2)[FRAME_WIDTH],
              (*raster3)[FRAME_WIDTH],(*raster4)[FRAME_WIDTH],
              (*temp)[FRAME_WIDTH];

       raster1 = new int[FRAME_HEIGHT][FRAME_WIDTH];
       raster2 = new int[FRAME_HEIGHT][FRAME_WIDTH];
       raster3 = new int[FRAME_HEIGHT][FRAME_WIDTH];
       raster4 = new int[FRAME_HEIGHT][FRAME_WIDTH];
       temp   = new int[FRAME_HEIGHT][FRAME_WIDTH];

       for(y=0; y<FRAME_HEIGHT; y++){
              for(x=0; x<FRAME_WIDTH; x++){
                     raster1[y][x] = 0;
                     raster2[y][x] = 0;
                     raster3[y][x] = 0;
                     raster4[y][x] = 0;
                     temp[y][x] = 0;

                     //combine two CD masks
                     temp[y][x] = (right_mask[y][x]) & (left_mask[y][x]);
```

```
		}
	}


	rightRaster(frame,temp,raster1);
	leftRaster(frame,temp,raster3);
	upRaster(frame, temp,raster2);
	downRaster(frame, temp,raster4);

	//combine resulting masks of rasters
	for(i=0; i<FRAME_HEIGHT; i++){
		for(j=0; j<FRAME_WIDTH; j++){
			final_mask[i][j] = (raster3 [i][j])| (raster4[i][j]);
		}
	}

	delete [] raster1;
	delete [] raster2;
	delete [] raster3;
	delete [] raster4;
	temp [] raster4;
}

//*************** final object generation from original frame *************
//this function superimposes the final object mask over the current
//frame and produces the final segmented object
void PostP::superImpose(int frame[][FRAME_WIDTH],int mask[][FRAME_WIDTH],
			int final_mask[][FRAME_WIDTH])
{
	register int i, j;

	for(i=2; i<FRAME_HEIGHT-2; i++){
		for(j=2; j<FRAME_WIDTH-2; j++){
			if(mask[i][j]>=1) final_mask[i][j] = frame[i][j];
			else final_mask[i][j] = 250;


		}
	}

}
//*****************************************************************************

//********************* end of PP class implementation *********************
```

Declaration

I, the undersigned, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been fully acknowledged.

Name: Eneyachew Tamir Signature: _____

Place: Addis Ababa

Date of submission: December 2007

This thesis has been submitted for examination with my approval as a university advisor.

Dr. Kumudha Raimond Signature: _____

Advisor's Name