In the name of God,
the most beneficent,
the most merciful

A Practical Vision System for the Detection of Moving Objects

by

Bijan Shoushtarian

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

Loughborough University
Department of Computer Science
June 2011

Acknowledgements

I am especially grateful for the financial support of the Ministry of Science, Research and Technology (MSRT) of Iran, the University of Isfahan, and Computer Engineering Dept. for my doctoral scholarship.

I dedicate this work to my wife, Afsaneh, and my children, Farzad and Farnaz, for their patience, sincere support and encouragement to complete my research and write my thesis. I also dedicate this work to my parents who provided me with invaluable support on all matters.

Table of Contents

Abstract

The main goal of this thesis is to review and offer robust and efficient algorithms for the detection (or the segmentation) of foreground objects in indoor and outdoor scenes using colour image sequences captured by a stationary camera. For this purpose, the block diagram of a simple vision system is offered in Chapter 2. First this block diagram gives the idea of a precise order of blocks and their tasks, which should be performed to detect moving foreground objects. Second, a check mark (✓) on the top right corner of a block indicates that this thesis contains a review of the most recent algorithms and/or some relevant research about it.

In many computer vision applications, segmenting and extraction of moving objects in video sequences is an essential task. Background subtraction has been widely used for this purpose as the first step.

In this work, a review of the efficiency of a number of important background subtraction and modelling algorithms, along with their major features, are presented. In addition, two background approaches are offered. The first approach is a 'Pixel-based' technique whereas the second one works at object level. For each approach, three algorithms are presented. They are called 'Selective Update Using Non-Foreground Pixels of the Input Image', 'Selective Update Using Temporal Averaging' and 'Selective Update Using Temporal Median', respectively in this thesis. The first approach has some deficiencies, which makes it incapable to produce a correct dynamic background. Three methods of the second approach use an invariant colour filter and a suitable motion tracking technique, which selectively exclude foreground objects (or blobs) from the background frames. The difference between the three algorithms of the second approach is in updating process of the background pixels. It is shown that the 'Selective Update Using Temporal Median' method produces the correct background image for each input frame.

Representing foreground regions using their boundaries is also an important task. Thus, an appropriate RLE contour tracing algorithm has been implemented for this purpose. However, after the thresholding process, the boundaries of foreground regions often have jagged appearances. Thus, foreground regions may not correctly be recognised reliably due to their corrupted boundaries. A very efficient boundary smoothing method based on the RLE data is proposed in Chapter 7. It just smoothes

the external and internal boundaries of foreground objects and does not distort the silhouettes of foreground objects. As a result, it is very fast and does not blur the image.

Finally, the goal of this thesis has been presenting simple, practical and efficient algorithms with little constraints which can run in real time.

Chapter 1

Introduction

1.1 Motivation

Vision is the primary sense of human beings. Using this sense, a large amount of information can be perceived. For example, by looking at a few images, a person may receive a great deal of valuable information which may not be obtained by other senses as easily. Even if the information of those images is written as a text, it may consist of a number of pages and still the information of the images is not conveyed well. On the other hand, it is possible that the capability of seeing the environment, i.e. the visual sense, is provided for a man-made system, such as a robot. In this way, it is observed that the robot's abilities, e.g. finding a correct route and transferring objects to specific places accurately, are extremely increased. Therefore, vision is not only the primary sense for human beings but also using visual information is extensively being developed in various areas. Computer vision, image processing, robotics, information and communication technology (ICT), etc are among the fields that make maximum use of visual information (Umbaugh, 1998).

The computational power of standard PCs has increased over the past decades due to advances in micro-electronics industries. Many companies offer very powerful PCs at low prices. These personal computers are capable to perform highly complex operations in a short period of time. Meanwhile, there are a large number of CCD (charge couple devices) cameras on the market which their prices continuously decrease. The decreasing cost of cameras has been a good motivation so that cameras are used more and more for gathering visual information. In this regard, many of them are now being installed in different places for a variety of purposes. These cameras produce a large amount of data in form of video or image sequences which should be analysed. Therefore, due to availability of powerful PCs and cheap CCD cameras on the market, the range of computer vision applications has been extended since several years ago. Video security and surveillance, traffic monitoring and transportation, quality control and medical imagery are some typical examples of computer vision applications in different fields (Umbaugh, 1998).

Image processing is a field by which human beings examine and process images. The output (processed) images are also created for the utilisation of human beings. However, computer vision is a field in which applications require a computer to process visual information such as video or image sequences directly. Nowadays, these two fields have become more and more interrelated so that many image processing techniques are being used as essential tools for various computer vision tasks (Umbaugh, 1998).

The goal of computer vision is to develop efficient and robust techniques for visual data acquisition, analysis, interpretation and understanding. Image acquisition is concerned with capturing a video sequence and converting into a sequence of digital images using a frame grabber. Digital images are then stored on a hard disk for off-line processing or in the main memory for applications which need on-line processing. Once a sequence of images are captured and stored in the memory, an application starts its analysis. Due to a large number of pixels in each image, working at pixel level usually has a low efficiency and requires a long processing time. It is often preferred that the pixels, which have common features such as similar colours, are grouped together to constitute regions (Umbaugh, 1998).

In many computer vision applications, separating the regions of interest (ROIs) from the rest of the video, i.e. a segmentation process is an important task. Often these ROIs correspond to foreground objects which may be moving. Detection and tracking moving foreground objects are fundamental and crucial processes for recognising mobile objects. These processes are used to infer a high-level description of mobile objects' activities and behaviours. However, the robustness and effectivity of video understanding process highly depends on earlier stages. Computing an accurate dynamic background and identifying foreground objects without their shadows correctly are examples of such earlier stages. Finding algorithms, capable to reliably extract foreground objects from visual information, is still a great challenge which needs more investigation and research. This is the underlying motivation for this thesis.

1.2 The problem statement and the thesis organisation

The main goal of this thesis is to offer a robust and effective algorithm for detection (or segmentation) of foreground regions (or objects) in indoor and outdoor scenes using colour image sequences captured by a stationary camera.

Image segmentation, in general, is a sophisticated, difficult and fundamental problem in image processing. Its difficulty is not only due to the complexity of the mechanisms used but also has an inherent ill-posed nature. By segmentation, regions of pixels, which have some type of homogeneity such as colour, texture or motion, are grouped together. However, since there is not a precise and unique definition for segmentation, different algorithms may yield different segmentation results (Kim et al., 2001).

As the basis of the research in this thesis, a block diagram of a simple vision system is explained in ***Chapter 2*** (Fig. 2.1). The block diagram consists of a number

of blocks including: dynamic background generation (or modelling), thresholding, connected componenets labelling, size filter, shadow detection and removal, boundary detection and boundary (silhouette) smoothing, etc. In fact, ***Chapter 2*** offers the preliminary and literature review for the blocks of this diagram.

Dynamic background modelling is the first stage of many vision systems because a large number of pixels of each input image of a video sequence are often background pixels. If all background pixels of each input frame are correctly identified, it speeds up the later processes such as detection and tracking of foreground regions. In fact, recognition and separation background areas have a key role in segmentation of foreground regions. In this regard, numerous algorithms for background modelling have been reported in the literature since more than twenty years ago. A lot of vision researchers who have offered new detection and tracking algorithms have introduced background modelling methods as well. An introduction to background modelling and a review of the most important background algorithms is offered in ***Chapter 3***. In addition, short descriptions of two background subtraction algorithms are given at the end of ***Chapter 3***. This is because those two algorithms are compared with the proposed background method in Chapter 6.

Detecting foreground objects correctly is an important issue in computer vision systems. The pixels of foreground objects (regions) can be determined using the pixels of the colour difference image. For easier and faster processing, often colour difference images are converted to binary frames by thresholding. As a result, threshold values should be chosen suitably so that foreground and background pixels are correctly separated and minimum numbers of noisy pixels are appeared in the binary image. ***Chapter 4*** deals with selecting optimum thresholds for colour difference image.

A major part of research in this thesis is devoted to dynamic background modelling for colour video sequences. For this purpose, two groups of background approaches are proposed. The first one is a 'Pixel-based' technique whereas the second works at object level. For each approach, three algorithms are presented. They are called the 'Selective Update Using Non-Foreground Pixels of the Input Image', the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median', respectively.

In ***Chapter 5***, the 'Pixel-based' approach for background modelling using the above methods is presented. The pixels of each input image are discriminated using a normalised *rgb* colour filter in two major groups, i.e. background and foreground pixels. Then 'Pixel-based' approach updates only background pixels based on one of the above methods. Besides, in order to check wethter a pixel is in motion or remains stationary for a period of time, a timer is assigned to each pixel of the image. Thus, a pixel's timer is incremented if it is considered as a foreground pixel. At the end of the chapter, it is concluded that these 'Pixel-based' methods are unable to produce a correct dynamic background frame for each input image. This is due to failure of colour filter and some difficulties concerning the timers of foreground pixels. The deficiencies of the 'Pixel-based' methods conduct us to seek an 'Object-based' approach.

Three selective methods for an 'Object-based' approach are presented in ***Chapter 6***. The methods of the second approach use an invariant colour filter and a suitable motion tracking technique, which selectively exclude real foreground objects from updating and return the remaining regions as background. Then the total initial and returned background pixels are upadated. The difference between three algorithms of the second approach is in updating method of the background pixels. It is shown that the 'Selective Update Using Temporal Median' produces an appropriate background image for each input frame. The advantages of this method are: it operates in unconstrained outdoor and indoor scenes. Also it is able to handle difficult situations such as removing ghosts and including stationary objects in the background image effectively. Very good results obtained on a number of image sequences confirm the effectivity of the new algorithm. Finally, the comparison between the mentioned method and two algorithms is given in ***Chapter 6***.

In ***Chapter 7***, a contour tracing algorithm using RLE (run-length encoding) data based on Quek's algorithm (2000) is utilised and implemented. In addition, a very effective silhouette smoothing method based on RLE data is proposed in this chapter. In practice, a complex combination of horizontal, vertical and diagonal one-pixel-width out-spikes may occur on the boundaries of foreground regions. In this case, the RLE smoothing method is simultaneously performed as the contour of a foreground region is traced. Once an out-spike is found on a segment of the contour, it is smoothed. Then tracing is continued by considering the smoothed segment. This RLE smoothing method called min-of two method only works on external boundary pixels of objects and has no concern with the other pixels. The min-of-two method also removes all one-pixel-width out-spikes by a single traversal of a region boundary. As a result, it is very fast and does not blur the image. Finally, conclusions and future work are discussed in ***Chapter 8***.

Based on the above paragraphs, it is observed that the focus of the research in this thesis is based on dynamic background modelling and RLE boundary smoothing of the foreground regions.

Many algorithms for low-level and mid-level processing of video sequences in computer vision have been offered. A lot of them are highly sophisticated and theoretical. In addition, once they are implemented, they don't have the desired performance or may not run in real time. On the other hand, in this thesis, the emphasis is on avoiding complex and theoretical algorithms. The goal has been on presenting simple, practical and effective algorithms with little constraints which can run in real time. The performances of the proposed methods are also compared with corresponding suitable algorithms in the literature.

Chapter 2

The Block Diagram of a Simple Vision System

2.1 Introduction

The block diagram of a simple vision system is explained in section 2.2 which is the basis of the research in this thesis. This chapter is also concerned with the literature review of majority of the blocks of the simple vision system.

2.2 A simple vision block diagram

Fig. 2.1 shows the block diagram of a simple (low-level/mid-level) vision system. A check mark (✓) on the top right corner of a block indicates that this thesis contains a research about it. For an easier reference, the blocks of this block diagram are numbered by adding a sequence number on their top left corners. The details of each block are given in the following subsections.

2.2.1 Converting a video film to a sequence of images

It is assumed that the camera is stationary. In some applications such as video surveillance, traffic monitoring, etc the camera may be installed at a suitable and usually high altitude where it can view public areas such as highways, airports, streets, car-parks, etc.

Cameras produce video signals which are then converted to sequences of digital images using a hardware (or software) frame grabber. Some cameras also produce digital image sequences directly (block number 1 in Fig. 2.1).

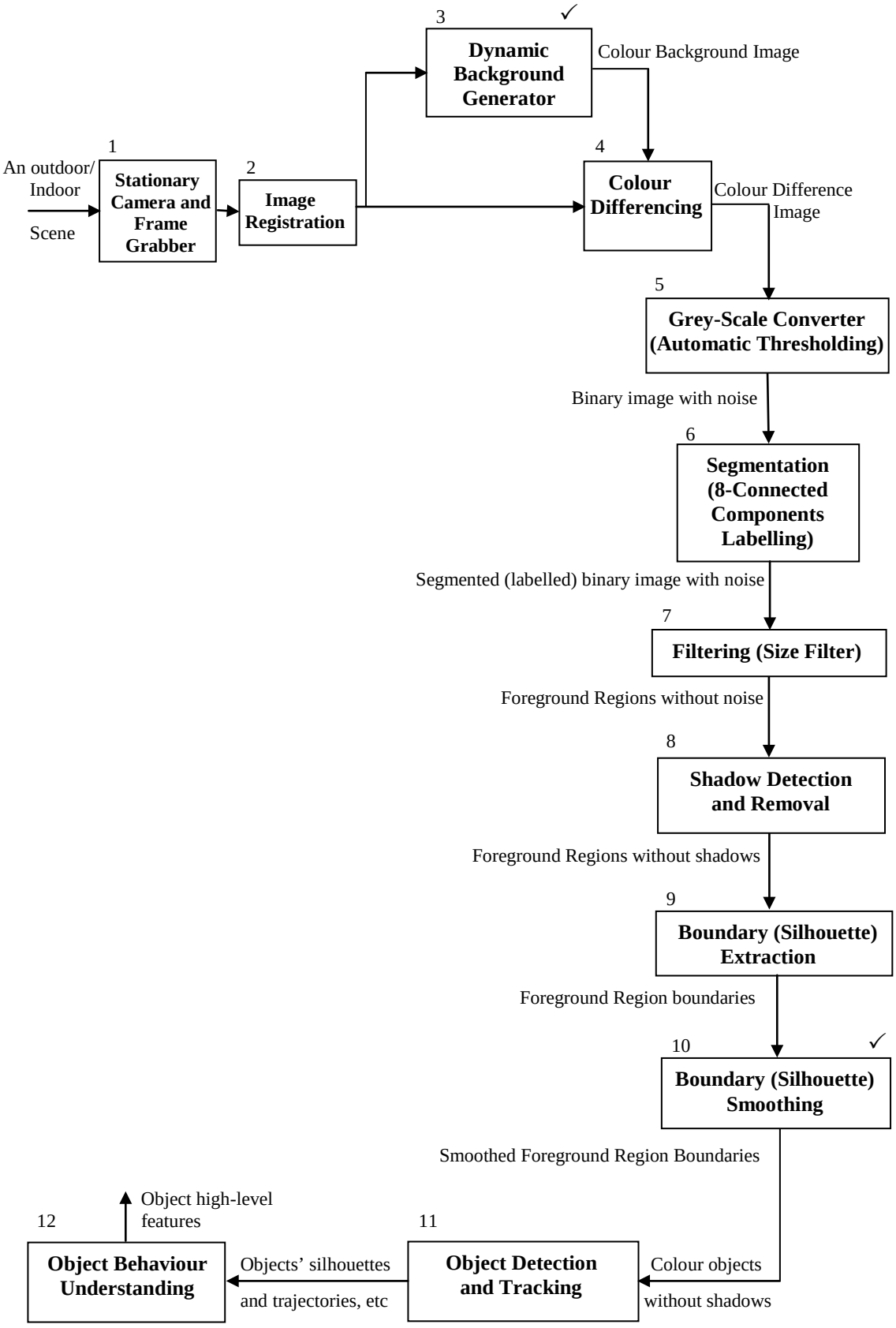


Fig. 2.1 – A simple block diagram for detecting and tracking objects

2.2.2 Image registration

It is very possible that the camera is shaken by the wind (in an outdoor scene) or by any other cause (in an indoor environment) and as a result the image plane changes slightly over time. This occurs when a stationary camera takes a video sequence from a scene but occasionally the camera has a small amount of motion.

In order to have the same image plane, which is necessary for later processes such as background generation, image differencing, etc, the input frames must be adjusted in a pre-processing stage. The process, by which the frames of an image sequence are aligned to same coordinate frame, is called *image registration*. Geometrical adjustments based on similarity, affine or projective transformations are often used as spatial transformations for image registration (Radke et al., 2005). For example, the images in a sequence can be corrected by translating them with respect to any image in the sequence or to some reference image such that their cross-correlation is minimised (Rosin and Ellis, 1995). Fortunately, image registration is a well-studied subject (Brown, 1992, Lavallee, 1995, Maintz and Viergever, 1998, and Zitova and Flusser, 2003) and software implementations (Ibanez et al., 2003) are available. Thus, after the frame grabber, the captured input frames are passed through an appropriate image -registration software so that the output images have the same coordinate system.

2.2.3 Background generation

Suppose there are a set of images of the same scene taken at different times. Sets of pixels in the last frame of a sequence, which are "*significantly different*" from their corresponding pixels in the previous frames, constitute the regions of change (Radke et al., 2005). Detecting regions of change in a video sequence captured from the same scene has attracted a great deal of attention in many computer vision applications. For this purpose, many spatial and/or temporal approaches have been proposed in the literature.

In video surveillance applications, change detection is closely related to the problem of background modelling. Here, the change detection problem is to determine background pixels prior to classifying the remaining pixels as foreground (i.e. changed) pixels which are then divided into different objects. In background modelling, all or a number of frames of a sequence may be utilised as the basis for deciding about changed areas (Radke et al., 2005). Thus one change detection method is concerned with producing background images. The question is how background images are generated using input frames?

A trivial approach for background estimation is where the reference image is obtained when the scene is static (i.e. there is no background motion). However, since there are variations in lighting conditions caused by changes of light level in an outdoor environment (e.g. due to changing position of the sun, clouds, etc), the reference frame gets out of date very soon. Thus, adaptive updating techniques should be applied to the background image in order to keep it up-to-date (Rosin and Ellis, 1995).

In a realistic situation, it may be impractical for a surveillance system to acquire a background image with no moving object (e.g. for a traffic surveillance system which monitors a street or a highway) and sometimes *stationary* objects are moved away from the scene (e.g. a parked car is taken out or a gate is opened and then is left opened) (Dawson-Howe, 1996). Thus, an adequate dynamic background modelling method is the solution to overcome these problems.

Background subtraction is recognised by the scientific community as a simple and common method for segmenting moving objects. Most background subtraction approaches consist of two steps: first a reference image/model is properly updated and then the current image is subtracted from the reference background image/model (Spagnolo et al., 2006).

Due to importance of dynamic background modelling in this thesis, the next three chapters are devoted to dynamic background generation/modelling for colour video sequences.

2.2.4 The colour difference image

Once a dynamic background image is obtained for each input frame, foreground regions can be detected using background subtraction, also called colour differencing. Background subtraction, which is performed in block number 4 (in Fig. 2.1), is obtained by taking the pixel-by-pixel absolute difference of the current image (I_i) and a background image (B_i) for each colour channel separately. The obtained frame is called colour difference image (D_i).

$$D_i(x, y) = | I_i(x, y) - B_i(x, y) | \qquad \text{(Eq. 2.1)}$$

where x and y are the row and column of the pixel and index i is the frame number. Eq. 2.1 is applied to red, green and blue colour channels individually. For example, Fig. 2.2a to Fig. 2.2c show an input image, its corresponding background and colour difference images, respectively.

2.2.5 Thresholding the colour difference image

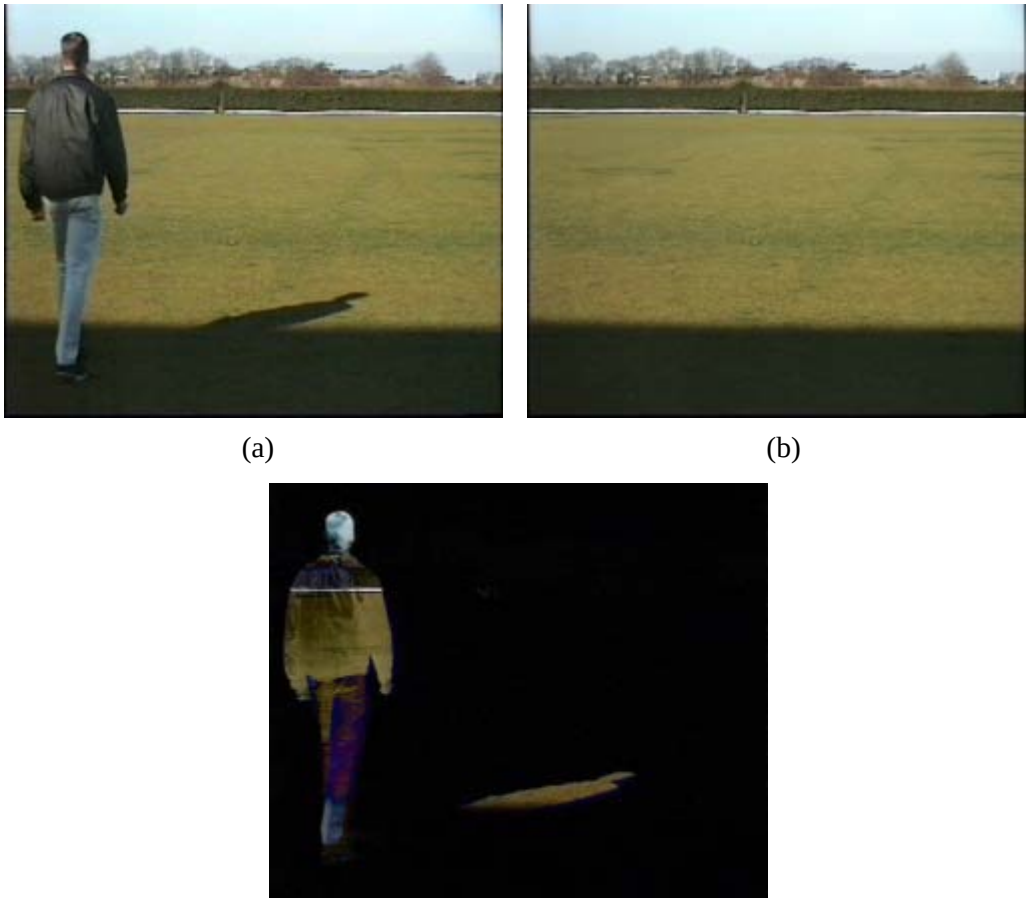
It is possible to determine the pixels of foreground regions using the pixels of the colour difference image. However, for later processes it is not easy and moreover it is very time-consuming if the decision is made based on three grey-scale (i.e. red, green and blue) images. For this purpose, it is better to convert the colour difference image to a binary frame. This requires thresholding the difference image by applying a grey-scale converter. After thresholding, foreground and background pixels of the thresholded (or binary) image are labelled by 1 and 0, respectively.

Thresholding is a simple and common technique by which the pixels of an image are divided into two dominant groups based on their grey-level intensities: the foreground (objects) and the background. The idea for this technique is that the grey

levels of the object pixels are significantly different from the grey levels of the background pixels (Sezgin and Sankur, 2004).

There are two main categories for thresholding techniques: global thresholding and local (adaptive) thresholding. In global thresholding, the foreground and the background of an image are separated based on a fixed threshold value applied to the whole image. Utilising a single value for all the pixels of an image makes the global thresholding a simple and effective tool which is sufficient in many cases (Solihin and Leedham, 1999).

Local thresholding methods use local information of the image and compute a threshold value for each pixel. Local thresholding is usually used when the image is unevenly illuminated. For this purpose, often a rectangular window is selected and local information such as intensity histograms of pixels (Chow and Kaneko, 1972, Eikvil et al., 1991, Taxt et al., 1989), maxima and minima of pixels (Bernsen, 1986) or mean and standard deviation of pixels (Niblack, 1986) in that window is considered. Then a threshold value is computed according to the region under consideration and is assigned to each pixel. Based on the size of the window, local thresholding methods need to perform a number of computations for the selection of the threshold value for each pixel. Therefore, local thresholding methods are much



(a) (b)

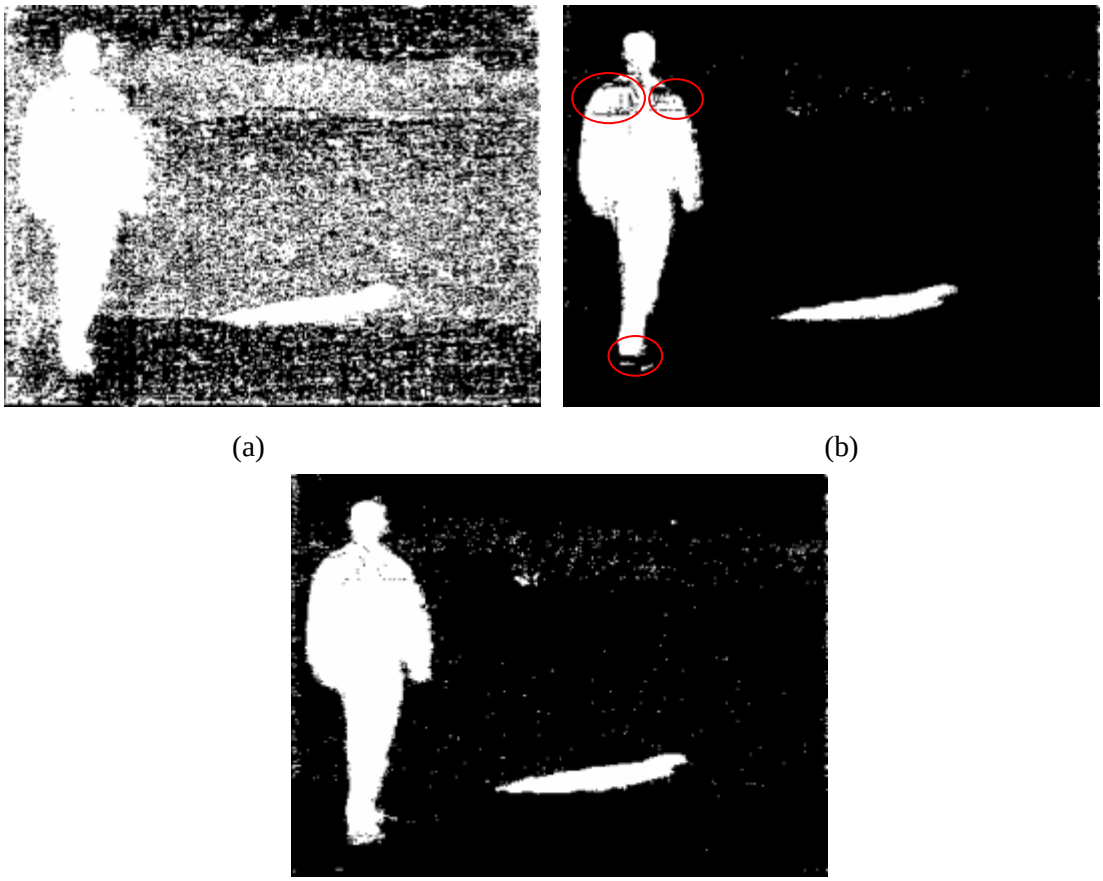
(c)

Fig. 2.2 – (a) Input image Fld249; (b) Background image Back-Img249; (c) Colour difference image Dif249.

slower than global thresholding techniques making it inefficient for many computer vision applications. Thus global thresholding is used in this work.

The magnitudes of pixel intensities of a difference image are the only information available for separating structural changes from the rest of the image. These magnitudes should be significantly high in order to detect such changes. Thus, the value of the threshold is of great importance.

The choice of a threshold τ should be slightly higher than the perturbations in pixel values of the difference image which are due to noise. If the value chosen for τ is too low, then spurious changes will be detected. If τ is too high, many pixels of the structural changes (i.e. the pixels of the objects) are removed as well. In this case, not only a large number of noise pixels are deleted, but also in the binary frame, some parts of the foreground objects are eroded (see Fig. 2.3).



(a) (b)

(c)

Fig. 2.3 – Thresholding the colour difference image Dif249; (a) Thresh249 with Red-τ = 8, Green-τ = 6, Blue-τ = 9, a large number of spurious pixels are visible in the thresholded image; (b) Thresh249 with Red-τ = 26, Green-τ = 24, Blue-τ = 30, some parts of the foreground object are eroded and the majority of the spurious pixels are removed; (c) Thresh249 with Red-τ = 12, Green-τ = 10, Blue-τ = 14, foreground objects are not eroded but a number of spurious pixels are visible in the thresholded image. It is important that the pixels of the foreground objects are not eroded. Then spurious pixels can easily be removed using a size filter.

Thresholds can be determined empirically (i.e. manually) or automatically. If the threshold is selected manually, it will indicate that an operator is involved in the system process. He should examine each difference image and interactively tunes a threshold value according to the scene characteristics. Obviously this approach is cumbersome and inappropriate for automatic computer vision applications which work on long image sequences. Therefore, for optimal threshold selection, a suitable automatic approach which depends on the scene content, the illumination conditions and the camera noise should be applied. Several automatic methods for threshold selection will be explained in Chpater 4 and an optimal threshold will be chosen (block number 5 in Fig. 2.1).

2.2.6 Connected components labelling

Connected components labelling is a fundamental task in many image processing and computer vision applications. The goal of this procedure is to extract and label various disjoint and connected components in an image for automating image analysis applications.

Once an image has been thresholded to produce a binary image, binary-1 pixels (i.e. pixels of foreground regions) are grouped together by a connected components labelling operator in order to obtain maximal connected regions. These binary-1 pixel components form candidate regions for representing objects in the binary image. The input to the labelling procedure is often a binary image but its grey-level version can also accept a grey-level image. The output of the labelling procedure is a symbolic image in which the pixels of each connected component are identified by a unique integer label (Haralick and Shapiro, 1992; Jain et al., 1995) (see Fig. 2.4).

Connected components labelling is an image transformation technique for grouping pixels into regions (or objects) (Haralick and Shapiro, 1992). The reasons for this transformation are as follows:

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

(a)

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

(b)

Fig. 2.4 – (a) A binary image; (b) Symbolic image produced from (a) by the connected components operator (Haralick and Shapiro, 1992).

1. For each individual colour pixel, there are a few properties which consist of its position, intensity and colour.

2. The properties of the individual pixels alone do not provide any useful and important information for the whole image.

3. A region consists of a number of pixels and has much a richer set of properties than a pixel. Shape, area, perimeter, position (i.e. the centre of gravity), average colour and the contour pixels are among the properties of a region.

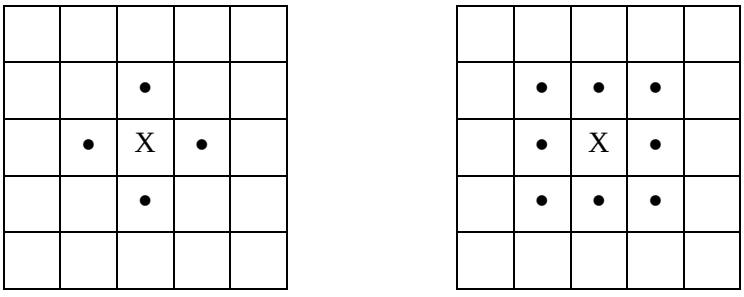
Connected components analysis of a binary image starts with scanning an image pixel by pixel (frequently from top to bottom and left to right). Two 1-pixels *p* and *q* belong to the same connected component if there is a path or a sequence of neighbour (or adjacent) 1-pixels among them. Then, based on the definition of connectivity between the pixels of a binary image (e.g. 4-connected or 8-connected neighbourhood – see Fig. 2.5), pixels are grouped into components.

There are several methods for implementing the connected components labelling operation including recursive, sequential and parallel. Recursive implementation procedure is time-consuming and requires a large stack for big images. Several parallel algorithms for labelling procedure have also been reported in the literature (Agrawal et al., 1987, Cheng et al., 1994, Ranganathan et al., 1995, Wang et al., 2003). However, due to emphasis on simplicity in this work, a simple and effective sequential method based on run-length encoding (RLE) has been used (Haralick and Shapiro, 1992, chapter 2).

After connected components labelling, connected foreground regions (or objects) are identified in the output symbolic image (block number 6 in Fig. 2.1).

2.2.7 Binary image representation and compression

In many computer vision applications, each frame of a colour video sequence contains a large number of pixels which need a great amount of memory. Due to



(a) (b)

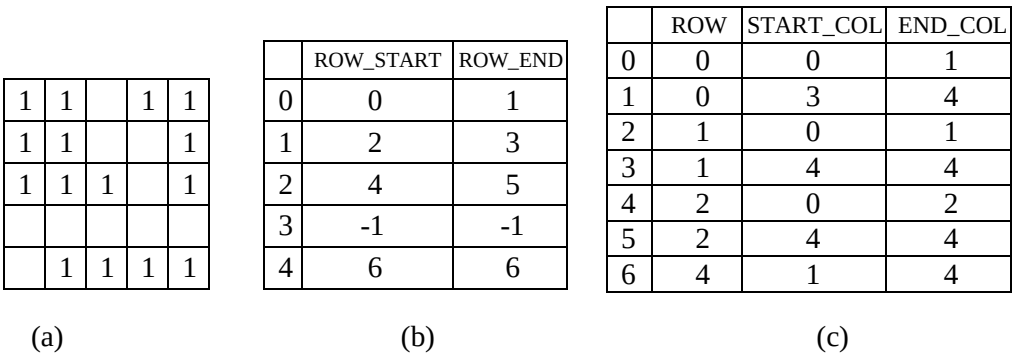
Fig. 2.5 – (a) Pixels • that are 4-connected to the centre pixel X; (b) Pixels • that are 8-connected to the centre pixel x; (Haralick and Shapiro, 1992).

consideration of computational time and memory space, it is often desirable to convert the colour difference image to corresponding binary image by thresholding. On the other hand, as the binary image may contain several regions and in order to extract features of foreground binary regions, they should be described in a form suitable for further processing. In this way, binary images are more conveniently and efficiently processed from the view point of computation time and required memory size. The main utilised approaches in this regard are based either on the characteristics of regions such as *run-length encoding* or region contours such as *chain codes* (Zingaretti, et al., 1998). Thus, after connected components labelling, foreground regions in a binary image are described by a suitable region representing approach. For this purpose useful information about both representations is given in this section.

2.2.7.1 Run-length encoding

Run-length encoding (RLE) is a compact representation of a binary image. RLE of a binary image represents each scan line of the image as a list of contiguous typically horizontal runs (sequence) of 1-pixels. RLE is commonly represented by two approaches. In the first one, each run is described by the location of the starting pixel of the run and the length of the run. In the second one (Fig. 2.6), the locations of the starting and the ending pixels are used (Haralick and Shapiro, 1992).

RLE has the advantage that a number of operations and image computations may be performed directly using its representation (Quek, 2000). 'And', 'Or', and 'Negation' are pixel-wise Boolean operations that can easily be performed on RLE data (Rosenfeld and Kak, 1982). Meanwhile, RLE can directly be utilised for geometric computations such as area and centroid (Klaus and Horn, 1986). Other operations such as connected components labelling can be accomplished using RLE representation as well (Rosenfeld and Kak, 1982). Also RLE is utilised in a number of applications such as image digitizers (Lindley, 1991), satellite image representation (Gonzalez and Wintz, 1987), image transmission, and in region representation for computer vision (Paul, 1986; Francis, 1990) and graphics (Newman, 1979).



1	1		1	1
1	1			1
1	1	1		1
	1	1	1	1

(a)

	ROW_START	ROW_END
0	0	1
1	2	3
2	4	5
3	-1	-1
4	6	6

(b)

	ROW	START_COL	END_COL
0	0	0	1
1	0	3	4
2	1	0	1
3	1	4	4
4	2	0	2
5	2	4	4
6	4	1	4

(c)

Fig. 2.6 – (a) A binary image, (b) its run-length encoding, and (c) Each run of 1-pixels is encoded by its row (ROW) and the columns of its starting and ending pixels (START_COL, and END_COL). In addition, for each row of the image, ROW_START points to the first run and ROW_END points to the last run of the row (Haralick and Shapiro, 1992).

2.2.7.2 Chain codes

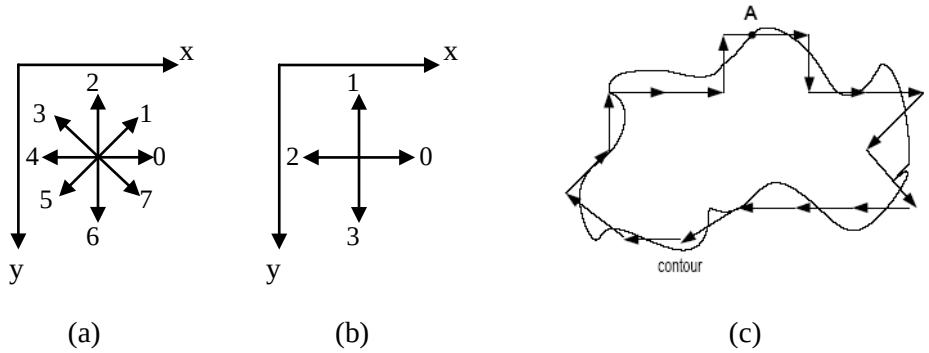
Once there are several foreground regions in a binary image, those regions can also be described by their boundaries. In *chain code* representation proposed by Freeman (1961), the contour of a region is represented by a start pixel address followed by a string of code words due to a sequence of movements around the region's border. The direction vectors between successive boundary pixels are encoded using 2-bit or 3-bit code words based on 4- or 8-connectivity (Fig. 2.7), respectively. Freeman chain codes for 4- and 8-connectivity are represented by the sets {0, 1, 2, 3} and {0, 1, 2, 3, 4, 5, 6, 7} as shown in Fig. 2.7a and Fig. 2.7b, respectively (Shih and Wong, 1999; Trimeche, 2000).

Regions/objects may have two types of boundaries: external and internal. The external boundary is the boundary which is surrounding the object while the internal boundary is the one surrounded by the object. External boundaries are traced counter-clockwise but internal boundaries are followed in a clockwise manner (Kim et al., 1988).

The chain code properties (based on 8-connectivity) are as follows:

- Horizontal and vertical directions correspond to even codes {0, 2, 4, 6} while diagonal directions are due to odd codes {1, 3, 5, 7}.
- Each code denotes an angular direction, i.e. $45i^{\circ}$ with $\{i \mid i = 0, 1, \ldots, 7\}$ contour-clockwise (or clockwise) from the positive *x*-axis.
- A region contour is completely described by the absolute coordinates (x, y) of the first contour pixel (e.g. top, leftmost) together with the chain code of the contour (i.e. a sequence of 3-bit code words).
- A change between two consecutive chain codes shows that the contour has changed direction. In this case, the corresponding point is defined as a corner.

Chain codes have the following advantages (Shih and Wong, 1999; Di Zenzo et al., 1996):



(a) (b) (c)

Fig. 2.7 – (a) 8-Directions Freeman chain code, (b) in 4-Directions, (c) Boundary pixel orientations: **A** 060057444543120020 and its 3-bit chain code: **A** 000 110 000 000 101 111 100 100 100 101 100 011 001 010 000 000 010 000 ((c) from Trimeche, 2000).

- A very compact region representation is provided by chain codes.

- Efficient coding is achieved by minimising the number of code words required to describe a boundary.

- Image processing and analysis are made easier.

- Chain codes are suitable for detecting features of a region such as sharp turns ('corners'), concavities, area, perimeter, centres, moments, projection and straight-line segments.

Chain codes have the following disadvantages (Levner, 2002):

- They are quite long.

- Chain codes are sensitive to noise, distortion and imperfect segmentation.

- They are not invariant to scale and rotation.

Some of the above problems can be alleviated. A coarser sample grid (i.e. larger pixels) can reduce the length of chain code at the expense of reducing precision. A coarser sample grid has also the advantage of compensating for the scale changes (Levner, 2002).

Due to above problems, some modified chain codes are also proposed as follows:

- Instead of the actual directions of the chain links, the derivative of the chain code is used in the *derivative chain code* method. 4-direction grid is used in this method and the directions are as follows (Levner, 2002; Trimeche, 2002):

  - 2 is the encoding for two successive links in the same direction.

  - 1 is the encoding for a convex corner.

  - 3 is the encoding for a concave corner.

- In order to have a rotation invariant code, instead of using the code itself, the first difference of the chain is utilised. Thus, a *difference chain code* is obtained by taking the difference of the adjacent elements. For example, if the original chain code is: 0, 0, 3, 0, 0, 3, 3, 3, 2, 1, 2, 2, …, then the difference chain code would be: 0, 3, -3, 0, 3, 0, 0, -1, -1, 1, 0, … (Levner, 2002).

- An alternative contour encoding method to the original chain code is called *crack code*. In this method, neither the contour pixels associated with the object nor the contour pixels associated with the background are used. Instead, the line, i.e. the 'crack' in between the object contour and the background contour is utilised. The crack code is similar to 4-direction chain code as illustrated in Fig. 2.8.

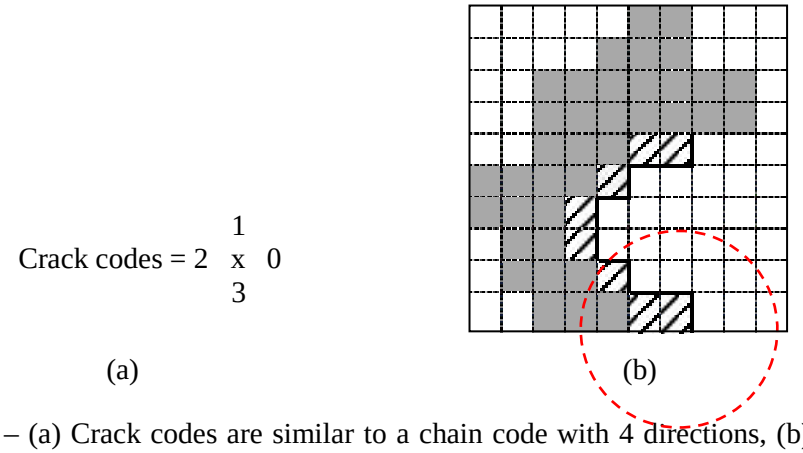


Fig. 2.8 – (a) Crack codes are similar to a chain code with 4 directions, (b) Contour pixels utilised in the chain code are diagonally shaded. The 'crack' is shown with the thick black line. The crack code is {3, 2, 2, 3, 2, 3, 3, 0, 3, 0, 0, 3}.

2.2.8 Size filter

After the connected components labelling process, connected foreground regions are identified in the corresponding input frame. Each region in the input frame is easily identified using the unique labels (integer values) of the output symbolic image. However, it is very probable that lots of spurious small regions and scattered pixels are also recognised. These spurious small regions usually do not correspond to any foreground region and are mostly due to camera motion, sensor noise, illumination variations, etc. So it is very important and necessary that such noisy regions are removed.

Two morphological operations, which can usually be used for removing noise, are 'erosion' and 'dilation'. For example, two successive erosions followed by one dilation operation may be used for cleaning the binary image. Morphological operators can not only serve as noise cleaning operators but also fill gaps or close the contours of the foreground regions (using 'opening' or 'closing' morphological operators). However, they have an adverse effect which makes their usages undesirable in some cases. In this work, morphological operators have not been used because they distort the shape of foreground objects.

There are many applications in which the objects of interest are greater than a minimum size of T_0 pixels. In such cases, after connected components labelling, the sizes of all regions can be computed easily. Then if a simple size filter is used, all components of size less than T_0 are removed (block number 7 in Fig. 2.1). In this case, the corresponding pixels of all these components are changed to 0. Thus, many small noisy regions are removed while the other components are preserved without any change in their shapes. Fig. 2.9 shows that the size filter is very effective in cleaning the noise while desirable components remain without any distortion.

The value of the threshold T_0 should be determined based on the application. If the foreground objects in a scene are very small, T_0 should be selected as a low value. On the other hand, if the foreground objects are large, the value of T_0 may

Fig. 2.9 – Thresh_Sizefilt249 is Thresh249 in Fig. 2.3c after applying the size filter. Almost all spurious pixels less than T_0 (e.g. $T_0 = 10$) have been removed from the thresholded image using a simple size filter. Meanwhile, the shapes of foreground objects have not been distorted.

be increased. In addition, the value of T_0 is not necessary to be fixed for the whole video sequence. In fact, T_0 can be adaptively set for each input frame based on several parameters. The number, the sizes and the amount of interactions between foreground objects can be among the parameters which may affect the correctness and exactness for the selection of T_0. Due to simplicity and effectivity of the size filter, finding an appropriate technique for selecting an adaptive threshold value T_0 for each input frame is of great importance.

2.2.9 Shadow detection and removal

For many computer vision applications, detection and tracking of moving objects is an essential task. A major problem, which occurs in both outdoor and indoor scenes, is that moving objects cast shadows over the surfaces which they move upon. Shadows may also be cast on surfaces close to the objects' paths (such as the walls).

Shadows represent a difficult phenomenon when detecting objects in outdoor and indoor scenes. In this case, shadows, which modify some characteristics of objects such as their shapes and colours, should be explicitly detected and effectively removed as otherwise they may be misclassified as objects or parts of objects.

Shadows can be considered from two points of view. First, they provide important information about the scenes in which they occur. This information may represent the type of light sources (point or wide) or the colour of the illuminentation unit, the characteristics of surfaces, the shapes and the relative positions of objects. Second, a cast shadow may be either attached to the object or completely separated from it. In

the first case, the shapes and the colours of segmented objects are distorted so that the information about the segmented object is not useful or reliable. As a result, the subsequent methods, which use the shape and colour of objects for their segmentation and classification, will fail. In the second case, the segmented shadow may be erroneously identified as an object in the scene not corresponding to any real object (Salvador et al., 2003). Thus, in many applications, segmenting and extracting moving objects will encounter serious problems when objects cast shadows.

For the above reasons, it is very important that shadows are accurately identified and segmented in order to recognise objects correctly. For this purpose, block number 8 (in Fig. 2.1) is dedicated to shadow detection and removal.

2.2.9.1 Cast shadow analysis

Shadows are usually classified as static or dynamic. Static objects such as buildings, parked cars, trees, etc produce static shadows while dynamic shadows are produced due to moving objects such as pedestrians, cars, trucks, etc (Nadimi and Bhanu, 2002). The cast shadow of a moving object is also called moving cast shadow (Stauder et al., 1999).

Shadows are produced due to total or partial occlusion of a light source in the scene by an object. A shadow consists of two parts: the *self-shadow* and the *cast shadow*. The part of the object, which is not illuminated by direct light, is called *self-shadow*. Meanwhile, the area projected on the scene in the direction of direct light is called cast shadow. Cast shadows are further divided into *umbra* and *penumbra*. The part of the cast shadow that the object has completely blocked the direct light is called an umbra. However, a penumbra is the part of the cast shadow where the object has partially blocked the direct light (Jiang and Ward, 1994). The umbra is darker and more easily detected than the penumbra. Meanwhile, it usually has a greater probability to be misclassified as a moving object (Prati et al., 2003). If the light source is a point, only umbrae are generated in shadows. However, both umbrae and penumbrae are produced due to an area light source.

In outdoor scenes, if the sky is cloudy, shadows are either weak (mostly penumbra) or non-existent. Meanwhile, there is no control over the illumination and little or no information may be available about the scene geometry (Nadimi and Bhanu, 2002). In indoor scenes, such as a lab, the illumination is often generated by an area light source. Thus, in indoor scenes, both umbrae and penumbrae may exist in cast shadows. On the other hand, in outdoor environments, the light source is often a far away point source (i.e. the sun). In addition, the distance between the objects and the background is negligible in comparison to the distance of illumination sources to objects (Nadimi and Bhanu, 2002). However, it is not logical to assume all cast shadows in outdoor scenes to be umbrae. Therefore, a shadow detection and removal algorithm designed to work in both indoor and outdoor environments should be able to detect the umbra. It may also detect the penumbra pixels in the cast shadow. Meanwhile, in an outdoor scene, the area of a penumbra may be very small

compared to an umbra; so the detection of a penumbra could be very difficult if the shadow has sharp edges (Salvador et al., 2004).

Shadows contain information about the shape and the relative positions of objects. They also provide cues about the characteristics of surfaces and light sources in the scene. However, tracking and recognition algorithms encounter difficulties when objects have moving cast shadows (Salvador et al., 2003).

Object merging, object shape distortion and even object losses (due to the shadow cast over another object) are the effects that shadows cause for the objects in images. Two main reasons concerned with shadow detection are as follows (Prati et al., 2003):

- Shadow pixels are detected as foreground pixels since their visual features (i.e. their brightness and colour) are significantly different from the corresponding background pixels.

- Shadows move along with their objects because the speed of objects and their cast shadows are the same. That is, shadows have the same motion as the objects casting them.

Thus, for accurate object detection and segmentation, effective shadow detection and removal algorithms are utilised.

2.2.9.2 Classification of shadow algorithms

In most shadow papers, a shadow model, which consists of a number of hypotheses, has been utilised. A summary of a shadow model is as follows (Prati et al., 2003):

- The light source is sufficiently strong.

- The light source is isotropically scattered within the object.

- There is a static reference image which is textured and planar.

- The object has perfectly matte (or Lambertian) surfaces.

For a good understanding about shadow algorithms, it is better to organise various algorithms based on a suitable taxonomy. Fortunately, Prati et al. (2003) has prposed an appropriate taxonomy for shadow algorithms.

Prati et al.'s two-layer taxonomy is shown in Fig. 2.10. In the first layer, the shadow algorithms are divided in two classes based on how uncertainty in the decision process is exploited. Based on this taxonomy, deterministic approaches use a yes/no (or on/off) decision to classify candidate shadow pixels. However, statistical approaches use probabilistic functions to decide on the class membership of the considered pixels. Statistical approaches usually use a number of parameters to determine class membership. Moreover, the parameter selection is a critical issue in

these approaches. For these reasons, statistical approaches are further divided into parametric and non-parametric methods. On the other hand, deterministic approaches are also divided into model based and non-model based methods depending on whether the decision process uses the knowledge of the model or not (Prati et al., 2003).

The second layer of the taxonomy in Fig. 2.10 shows the types of features that are exploited in the shadow detection process. These features are based on the following domains (Prati et al., 2003):

- Spectral: Approaches may use different spectral features, i.e. grey level or colour information.
- Spatial: Spatial information at a region level instead of pixel level may be used by some approaches to improve the results.
- Temporal: Temporal redundancy information may be exploited by some methods to integrate and verify the results obtained.

In another taxonomy presented by Salvador et al. (2004), shadow detection techniques have been classified into two groups: model-based and property-based approaches. A priori knowledge of the geometry of the scene, the objects, and the illumination is used by model-based techniques while features such as geometry, brightness or colour of shadows are used by property-based approaches. Model-based techniques are designed for specific applications such as aerial image understanding (e.g. Irvin and McKeown (1989); Wang et al. (1991); Bejanin et al. (1994)) and video surveillance (e.g. Koller et al. (1993); Sonoda and Ogata (1998); Yoneyama et al. (2003)). Sets of geometric features such as edges, lines and corners are used by model-based techniques to match with 3D object models. Obviously,

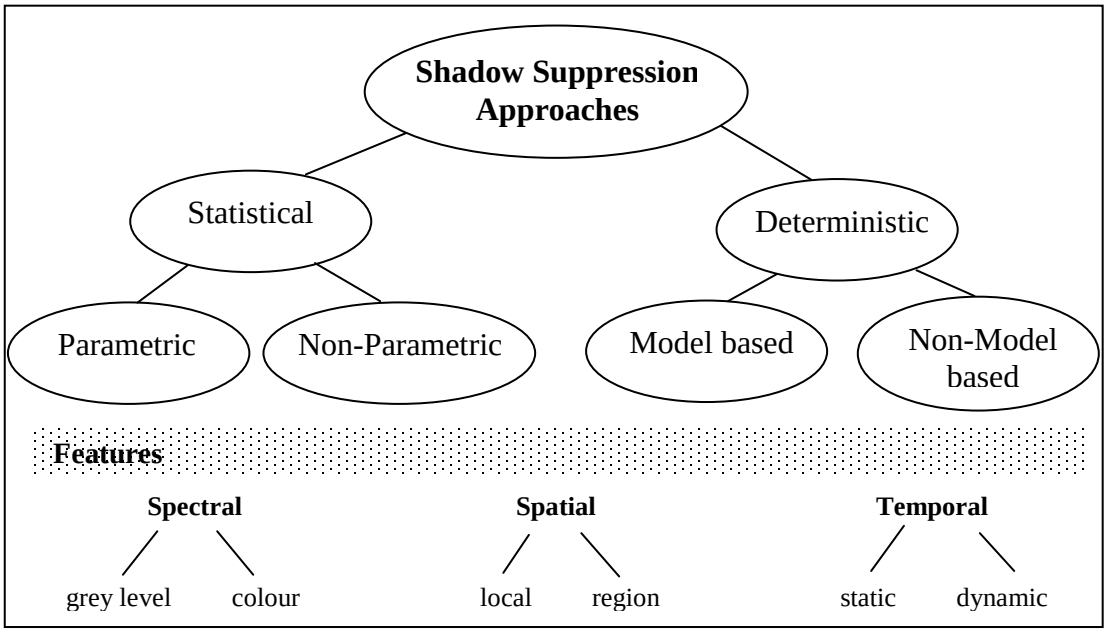


Fig. 2.10 – Shadow detector approaches taxonomy. (Prati et al., 2003).

these techniques are successful when dealing with simple objects and are only applied to specifically designed schemes. On the other hand, property-based techniques overcome these limitations by utili-sing spectral and geometrical features of shadows. Examples of these features are: luminance, chrominance, gradient density information, edges, texture information, colour ratios, invariant colour spaces, etc and some combinations of these features.

2.2.9.3 A review of a number of recent shadow papers

A review of recent shadow papers (from 2007 to 2010) is given in this section as follows:

Zha et al. (2007) proposed a shadow detection algorithm using normalised rgb for colour video sequences. They offered a uniform framework to eliminate cast shadow by energy minimisation. At first, moving objects are detected by background subtraction. Shadow candidates can be determined by pre-processing moving objects according to the shadow physical property. By comparing shadow and foreground points in the current input frame with their corresponding points in the background image, colour information, texture information and temporal-spatial coherence are obtained, which are then combined in a probability framework. Next, the maximisation of posterior probability will convert to an energy function based on Gibbs energy. Finally, moving cast shadows and foreground objects are segmented accurately by the minimisation of the energy function based on binary graph cuts.

The algorithm has the following advantages (Zha et al. 2007):

- The proposed method can work well in both indoor and outdoor scenes.
- In comparison with classical methods, it has good performance both in accurate shadow and foreground detections.

The disadvantage of the algorithm is (Zha et al. 2007):

- The parameters of the energy function are sensitive to different scenes and their values are determined empirically. They should be adjusted dynamically for better adaptability in different environments.

Zhang et al. (2007) proposed a moving cast shadow detection algorithm using ratio edges. Edge information can be used for shadow detection because edges in an image do not change with the illumination condition. Ratio edge is computed by the ratio of the intensity of one pixel to the intensity of its neighbouring pixels. It can be proved that ratio edge is illumination invariant. Then in ratio edge domain, background subtraction is performed. Shaded background areas are good candidates for shadow detection. Thus, in such areas, the distribution of the normalised background difference of ratio edge is analysed and is approximated to be a χ^2-distribution. Then, automatic shadow detection is performed by a significant test. Meanwhile, intensity constraint and geometric heuristics are imposed to further refine the detection results. Finally, shadow intensity ratio, which is defined as the ratio of the

intensity of the shaded image to the intensity of the background image, is estimated by implementing an iteration strategy. The iteration process can detect shadows quickly and correctly.

The algorithm has the following advantages (Zhang et al., 2007):

- In comparison with similar state-of-the-art methods, the algorithm is much simpler but more effective.
- It can detect moving cast shadows automatically and robustly.
- It has a better overall detection rate in comparison with similar methods.

The disadvantage of the algorithm is (Zhang et al., 2007):

- For updating the shadow intensity ratio Φ, a parameter λ needs to be set for each image sequence manually. Improper selection of λ severely affects shadow detection and shadow discrimination rates.

Fang et al. (2008) proposed a method for segmenting moving vehicle cast shadows in traffic surveillance images. In order to identify candidate shadow regions, an initial hypothesis is tested using a moving object detection algorithm. Then, the method exploits spectral and geometrical properties of shadows, the relationship between a point in shadow region and space position, and vehicle shape to verify this initial hypothesis. For segmenting the boundary between self-shadow and cast shadow, an occluding function is defined which is based on geometrical property of shadows. Meanwhile, a multi-resolution wavelet transform is used to detect the feature points of the occluding function.

The advantages of the algorithm are (Fang et al., 2008):

- The method does not need any camera calibration or a priori information regarding the scene.
- There is no restriction on the colour difference between vehicle and background.
- It has no knowledge in advance about the illumination direction.
- For improving the speed and robustness of the algorithm, the features of the occluding function are detected using 1D wavelet transform. As a result, the suggested method is real-time even on common PCs.

The disadvantages of the algorithm are (Fang et al., 2008):

- In video sequences of high traffic, the assumption that background is stationary may not lead to correct detection of foreground objects and their cast shadows.
- The method is suitable for simple scenes and may fail to detect and segment

shadows in complicated environments.

- The algorithm considers the first image of a sequence as the first background frame. In video sequences of high traffic, this assumption may not be valid. Thus, it may affect the accurate detection of foreground objects and their cast shadows.

Joshi and Papanikolopoulos (2008) proposed an adaptive technique for detecting moving shadows in colour video sequences. For extracting useful features, at first, a background model using mixture of Gaussians is obtained. Next, for each pixel in the detected foreground regions (or foreground mask), features based on illumination, colour and edges are extracted. Then, a learning technique is used, which employs support vector machines (SVMs) and a co-training method to train the algorithm with a small set of labelled image data for shadow detection. In fact, two classifiers (SVMs) are trained in this algorithm using two different feature sets on the initial labelled data. For detecting shadows on the unlabelled image data, each classifier is deployed at each round, which chooses from each class the example that it can label most confidently. This process is iteratively repeated for a fixed number of rounds or until all original unlabelled data are labelled.

The advantages of the algorithm are (Joshi and Papanikolopoulos, 2008):

- In comparison with other supervised approaches, the algorithm requires a small quantity of human labelled image data.

- The algorithm employs a semi-supervised learning technique which makes the model adaptive to changing scene conditions and gives better classification accurately at the same time.

The algorithm has the following disadvantages (Joshi and Papanikolopoulos, 2008):

- Due to high cost of manual labelling, providing the initial labelled data set may be a time-consuming and difficult task.

- If the size of the initial labelled data set is large, the algorithm requires a long time for training.

- It is hard to select an appropriate size of training data which provides good accuracy values for any new video sequence.

- Co-training needs to run online as new data become available. If the number of co-training rounds is large, the algorithm will fail to run in real-time.

Carmona et al. (2008) proposed a new approach for segmenting moving objects based on blob-level knowledge. The algorithm considers a number of categories in each frame including: real moving objects, shadows, ghosts, reflections, fluctuation or background noise region(s). The idea of the algorithm is to process each pixel differently depending on to which mentioned category it belongs. In this regard, for each pixel in the input frame at time t and position (x, y), the relation existing

between its RGB vector, i.e. $I_t^{(x,y)}(r, g, b)$ and its corresponding background vector, i.e. $B_t^{(x,y)}(r, g, b)$ is characterised with the value of the angle that they form, i.e. $\theta_t^{(x,y)}$ and the magnitude of difference of their modules in absolute value, i.e. $\Delta_{t\text{mod}}^{(x,y)}$. For this new two-dimensional space named angle-module space, a rule called angle-module rule is defined whose application produces an initial approach to the final foreground map. Next, reflections are eliminated from the foreground map by applying a reflection filtering operator. Similarly by applying shadow, ghost and background noise filters, respectively, moving foreground blobs are obtained as the final stage of the segmentation process.

The algorithm has the following advantages (Carmona et al., 2008):

- The proposed method is adaptive since it can update both the background and threshold models.
- The results obtained confirm the robustness of the method as it can operate in different indoor and outdoor scenes.
- It has low computational cost and as a result, it is a real-time algorithm.

The disadvantage of the algorithm is (Carmona et al., 2008):

- The algorithm considers the first input image as the first background frame. Thus, ghosts will appear in the background frame for the first $C_{\max}$ = 48 images. Therefore, the ghost elimination filter fails to remove ghosts for about two seconds (by assuming 25 frames per second).

Huang and Chen (2009) proposed an online statistical learning approach to model the background appearance variations under cast shadows. By assuming constant ambient illumination and direct light sources, at first, normalised spectral ratio is derived as the background surface invariant colour features based on the bi-illuminant dichromatic reflection model (Maxwell et al., 2008). The normalised spectral ratio remains constant independent of different background surfaces and illumination conditions. Then, the colour features extracted from all moving pixels are modelled using a single Gaussian mixture model (GMM). In order to differentiate cast shadows having similar colours to background, a pixel-based GMM is utilised, which describes the gradient intensity distribution for each pixel. The pixel-based GMMs are updated using the confidence predicted from the global GMM through confidence-rated learning to accelerate convergence rates.

The algorithm has the following advantages (Huang and Chen, 2009):

- It can learn model parameters very fast in an unsupervised manner and adapt to illumination conditions or environment changes.
- The method is robust to scenes with few foreground activities and videos captured at low or unsteady frame rates.

The algorithm has the following disadvantages (Huang and Chen, 2009):

- Due to high computational load, the method will run at very low frame rates.

- If the local shadow model is learnt following conventional Gaussian mixture learning method, the algorithm's model may still not be built due to the long training time and disturbance by foreground objects. As a result, it may affect the algorithm's performance or the algorithm may even fail in such circumstances.

Tian et al. (2009) proposed a method to extract shadows from a single outdoor colour image. A tricolour attenuation model (TAM) describes the attenuation relationship between shadow and non-shadow background. The algorithm's idea is that for every pixel in the image, if the minimum attenuated channel is subtracted from the maximum attenuated channel, the results in shadow regions will be lower than the results in non-shadow regions. This is based on the mechanism of image formation. The parameters of the TAM are determined using the spectral power distribution of daylight and skylight, which are estimated by employing Planck's blackbody irradiance theory. Finally, based on the proposed TAM, a multi-step shadow extraction method detects shadows in the image.

The advantages of the algorithm are (Tian et al., 2009):

- Unlike most other methods which are suitable for video sequences, the algorithm can detect shadows from only a single image.

- The algorithm is not designed for specific applications and can automatically extract shadows even in complex outdoor scenes.

- The algorithm does not need any prior knowledge about the scene or the light source and is completely data-driven.

The disadvantages of the algorithm are (Tian et al., 2009):

- It extracts all types of shadows in an image. Thus, it cannot distinguish cast shadows from self-shadows and shadows of stationary objects such as buildings and trees. Thus, the algorithm cannot be used for removing moving cast shadows in image sequences.

- The method will fail to detect shadows in sunrise and sunset because at these times the CCTs (correlated colour temperatures) of sunlight and skylight are very different from the CCTs adopted in the algorithm.

Jung (2009) presented a method for background subtraction and shadow removal for greyscale (monochromatic) video sequences. An α-metrically trimmed mean are used as a robust estimator in the training stage to model the background. Besides, in order to evaluate the spread of noise around the actual background value, the mean absolute deviation (MAD) is used as a robust scale estimator. Then, foreground pixels are detected in the evaluation stage by local spatial coherence to minimise the

occurrence of isolated foreground pixels. A combination of a statistical model and expected geometrical properties based on the relations of pixel ratios within small neighbourhoods are utilised for shadow identification and removal. Finally, for removing isolated foreground pixels and residual noise, a morphological post-processing scheme is used.

The algorithm has the following advantage (Jung, 2009):

- It uses a simple and fast foreground test, which effectively detect foreground objects and remove shadows in greyscale video sequences.

The disadvantages of the algorithm are (Jung, 2009):

- It is not suitable for colour image sequences as it does not use colour cues.
- The method focuses on identifying weak shadows in indoor scenes. In outdoor environments, strong shadows in sunny days are misclassified as dark objects.
- Homogeneous foreground objects, which appear in front of homogeneous portions of the background, may be erroneously classified as shadow pixels.
- If foreground objects have distinct colours in comparison with the background but the luminance component of these objects are similar or exactly the same, the proposed algorithm, which relies only on luminance, will fail.

Johansson et al. (2009) proposed a method that combines shadow detection and a 3D box model including shadow simulation, for estimation of size and position of vehicles. For classifying each pixel into background or foreground, a GMM is used to estimate the colour distribution over time. If the colour of a pixel is unlikely to belong to the distribution, that pixel is considered as foreground, which is further classified into foreground/shadow/highlight categories. If the colour of a pixel lies in a cylinder region between the black (the origin) and any of the centre colours of the background Gaussians (see Fig. 2.11), the pixel is classified as a shadow pixel. The shadow cylinder lies between 0.3 and 0.95 of the Gaussian centre with radius 15 (based on RGB colour space with range [0-255]). Highlight pixels are defined as pixels that do not belong to foreground objects and are brighter than the average background colour. If a pixel's colour lies in a cylinder between 1.05 and 1.5

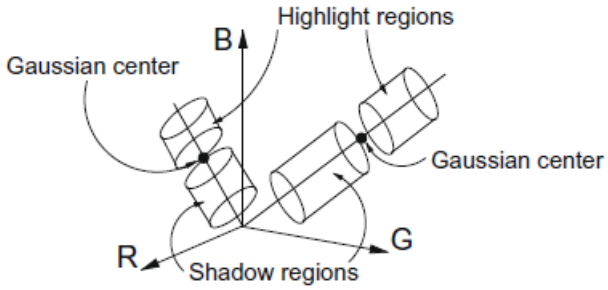


Fig. 2.11 – Illustration of the shadow/highlight classification in the RGB color space (Johansson et al., 2009).

above the Gaussian centre with radius 30 (Fig. 2.11), that pixel is classified as a highlight pixel. In addition, a similarity measure is defined between a simulated image of a 3D box, including the box shadow, and a captured image that is segmented into background/foreground/shadow regions. Then, an optimisation procedure is utilised to find the optimal box state, which is based on the similarity measure.

The algorithm has the following advantages (Johansson et al., 2009):

- It is shown in a number of examples that the combination of shadow detection and shadow simulation can improve the performance compared with only using either method. The improvement is more evident in cases where the shadow detection or shadow simulation is inaccurate.

- The 3D box optimisation including highlight detection and spatial window is used for predicting heading and refining box size estimates.

The disadvantages of the algorithm are (Johansson et al., 2009):

- A fundamental problem is concerned with the time window for learning of the GMM background model. Tuning the time window to suite all varieties of traffic density and light conditions is a difficult task. In fact, heavy traffic and varying light conditions will cause slower convergence of the background model since the model is not updated in foreground/shadow/highlight regions.

- The vehicles, which temporarily remain stationary for a long period of time, will eventually disappear and become part of the background model. Once they resume their course, they may leave ghost vehicles behind them in the background. Thus, it may affect the correct operation of the algorithm.

Choi et al. (2010) proposed an adaptive shadow estimator to detect and eliminate the shadow of a moving object automatically while adapting to varying illumination of the environment. First, a GMM is used to produce the background image. After background subtraction and in order to discriminate the shadow and the moving object, the algorithm uses a cascading of chromaticity difference estimator, brightness difference estimator, and local relation estimator. After these estimators, the set of moving pixels is divided into the candidate set of object pixels and the final candidate set of shadow pixels. Finally, a spatial adjustment is performed as the final step; i.e. a small object pixel region inside the shadow region is considered as a shadow and a small shadow pixel region in the object region is considered as a moving object.

The advantages of the algorithm are (Choi et al., 2010):

- The thresholds of its estimators are computed automatically. In addition, the algorithm does not need an additional training step. Thus, it rapidly adapts to variation in the environment.

- It is fast enough to operate in real-time. Thus, it is appropriate for surveillance applications.

- It outperforms the existing adaptive methods and has a superior performance over the existing learning-based and manual setting methods.

The algorithm has the following disadvantages (Choi et al., 2010):

- Once the algorithm starts, if the illumination changes quickly, the GMM background model needs time to adapt to varying environment. During this period, the background and the moving object set are invalid and as a result, shadow estimation is meaningless.

- If there are two or more light sources, e.g. in an indoor environment, the algorithm's performance drops substantially.

The recent shadow papers presented in this section, which are classified based on Prati et al.'s (2003) taxonomy, are shown in Table 2.1.

Statistical Parametric				Statistical Non-Parametric			
Paper	Spectral	Spatial	Temporal	Paper	Spectral	Spatial	Temporal
Zha et al. (2007)	C	L & R	D	Fang et al. (2008)	C	R	D
Zhang et al. (2007)	C	L & R	D	Joshi and Papanikolopoulos (2008)	C	L & R	D
Jung (2009)	G	L & R	D	Huang and Chen (2009)	C	L	D
Johansson et al. (2009)	C	L & R	D	Choi et al. (2010)	C	L & R	D
Deterministic Model-based				Deterministic Non-Model-based			
Paper	Spectral	Spatial	Temporal	Paper	Spectral	Spatial	Temporal
				Carmona et al. (2008)	C	R	D
				Tian et al. (2009)	C	L & R	S

Table 2.1 – Classification of a number of recent shadow algorithms (G = Grey-level, C = Colour, L = Local/Pixel-level, R = Region-level, S = Static, D = Dynamic).

2.2.9.4 Performance evaluation of shadow algorithms based on quantitative and qualitative metrics

Various shadow detection and removal algorithms can be systematically evaluated based on quantitative and qualitative measurements. For this purpose, a number of systematic evaluation metrics are proposed in the literature as follows:

1. The following two quality measures must be identified (Prati et al., 2003):

- *Good detection*: the probability, which a shadow point is misclassified, should be low. This corresponds to minimising false negatives (*FN*), i.e. the number of shadow points classified as background or foreground.

- *Good discrimination*: the probability, which a non-shadow point is misclassified as a shadow point, should be low. This one is also equivalent to minimising false positives (*FP*), i.e. the number of foreground or background points detected as shadows.

2. Medioni (1999) proposed two metrics for the evaluation of moving object detection. They are *Detection rate* (*DR*) and *False Alarm Rate* (*FAR*) defined as:

$$DR = \frac{TP}{TP + FN} \quad ; \quad FAR = \frac{FP}{TP + FP} \qquad \text{(Eq. 2.2)}$$

where *TP* is the number of true positives (i.e. the shadow points correctly identified), *FN* and *FP* are defined as before. However, for shadow evaluation, *DR* and *FAR* are not suitable metrics since it is not specified whether a detected shadow point belongs to a foreground object or to the background. For example, false positives belonging to background do affect neither the object detection nor the object shape. So in this case, shadow detection cannot be used for improving moving object detection since only *DR* (but not *FAR*) is problematic (Prati et al., 2003).

3. Prati et al. (2003) modified the *DR* and *FAR* metrics and proposed the *shadow detection accuracy* η and the *shadow discrimination accuracy* ξ defined as:

$$\eta = \frac{TP_S}{TP_S + FN_S} \quad ; \quad \xi = \frac{\overline{TP_F}}{TP_F + FN_F} = \frac{TP_S - FP_S}{TP_S + FN_S} \qquad \text{(Eq. 2.3)}$$

where the subscripts S and F correspond to shadow and foreground points, respectively. $\overline{TP_F}$ is equal to TP - FP. TP is the number of ground-truth points of the foreground object that were correctly detected. Meanwhile, FP is the number of points detected as shadows but belonging to foreground objects. In addition to the above quantitative metrics, *robustness to noise*, *flexibility to shadow strength, width and shape*, *object independence*, *scene independence*, *computational load*, and *detection of indirect cast shadows and penumbra* are considered as qualitative measures in their evaluation.

4. Tattersall and Dawson-Howe (2003) accept the second quantitative metric (i.e. *shadow discrimination accuracy* ξ) of Prati et al. (2003). However, they argue that this metric cannot be used to evaluate how well the foreground pixels were classified because it does not take into account the shadow pixels that have been incorrectly classified as foreground pixels. For this purpose, they propose a new metric that is concerned solely with how accurately foreground pixels have been identified as:

$$\varphi = \frac{TP_F - FN_S}{TP_F + FN_F} \qquad \text{(Eq. 2.4)}$$

where TP_F and FN_F are the same as before and FN_S corresponds to all the pixels that should be foreground but were classified as shadows.

5. Cavallaro et al. (2005) presented an objective evaluation which is performed with respect to ground-truth segmentation. In fact in their approach, the evaluation of shadow segmentation is done through the evaluation of video object segmentation. For obtaining an objective evaluation, the deviation of segmentation mask with respect to ground-truth segmentation is considered. They define two types of errors in each frame of sequence *n*, i.e. false positives $\varepsilon_p(n)$and false negatives $\varepsilon_n(n)$. Incorrectly detected pixels belonging to the object mask are defined as false positives. Meanwhile, pixels belonging to the object but not detected are false negatives. *card(C(n))* is defined to represent the number of pixels detected as object pixels at frame *n* and similarly $card(C_g(n))$ represents the number of pixels belonging to the ground-truth. Then they define deviation from the reference segmentation as:

$$\varepsilon(n) = \begin{cases} 0 & \text{if } Card(C(n)) = 0 \ \wedge \ Card(C_g(n)) = 0 \\ \dfrac{\varepsilon_n(n) + \varepsilon_p(n)}{Card(C(n)) + Card(C_g(n))} & \text{otherwise} \end{cases} \qquad \text{(Eq. 2.5)}$$

where $\varepsilon(n)$ is in [0, 1]. The spatial accuracy of the segmentation result is then quantified as:

$$v(n) = 1 - \varepsilon(n) \qquad \text{(Eq. 2.6)}$$

that its values are in [0, 1]. If $v(n) = 1$ then there is a perfect match between segmentation results and ground-truth.

2.2.9.5 Limitations of shadow algorithms

The main limitations of many shadow algorithms, which may cause their failure in some situations, are as follows (Porikli and Thornton, 2005):

- A shadow algorithm may fail when pixels of foreground objects are darker than the background and have a uniform gain with respect to the reference surface they cover. This happens when the algorithm only exploits luminance based criteria.

- Shadow algorithms based on geometrical model heavily depend on the viewpoint of the camera and the shapes of objects.

- Achieving robust shadow elimination for a wide range of conditions may not be possible just by using several predefined (or fixed) parameters.

- Most shadow algorithms are unable to adapt themselves to different types of shadows, e.g. light shadow (due to ambient light source) or heavy shadow (due to strong spot lights). The main reason for their failure is due to using fixed thresholds for various conditions.

Therefore, a powerful shadow algorithm should be dynamic so that it can be updated with each frame. For example, if the lighting condition (and as a result, shadow properties) changes, it can dynamically adapt itself to new condition automatically. Unfortunately, the main limitation of such dynamic shadow algorithms is their high computational loads which hopefully run in real-time by appearing more powerful PCs in near future.

2.2.10 Boundary extraction

Locating the boundaries of objects is a fundamental task in a variety of image analysis and computer vision applications. This process is also equivalent to an image segmentation method. If the boundaries between objects are found, objects are indirectly defined. Boundaries, which characterise the shape of an object, can be obtained by linking edges. Once they are specified, geometrical features of objects (or regions) such as size and orientation are easily computed (Umbaugh, 1998).

Once connected foreground regions are identified in the thresholded binary image, lots of spurious small regions and scattered pixels are also recognised which are due to noise. Thus, such noisy regions can be removed using, for example, a size filter to clean up the binary image. In addition, for correct recognition of objects, the cast shadows of foreground objects should also be removed using an appropriate method as explained in section 2.2.9.3.

After removing the cast shadows of objects, the silhouettes of all of the objects are obtained using a boundary extraction method. The shape, location, and orient-ation of silhouettes of regions are the primary information content within a binary image. Such information can be obtained by the knowledge of the boundaries of the regions (Capson, 1984).

The shape of objects often constitutes a major feature in different areas such as in visual databases (image, graphics and video) since it contains meaningful semantic information about the associated visual object (Xiao et al., 2001). In a user survey presented by Lambert (et al., 1999) concerning cognition aspects of image retrieval, it was concluded that users are more interested in retrieval by shape than by colour or texture. On the other hand, in computer vision, one of the most frequently used approaches for image segmentation is based on the extraction of the contours of the objects. The main reason for this selection is that most of the information about the shape of a 2D object is found in its contour (Iannizzotto and Vita, 2000). Also representing the shape of an object by its contour corresponds to the way that humans perceive objects. Due to this reason, the visual system of human beings concentrates on edges and ignores uniform regions (Hildreth, 1983).

An important feature of the shape of an object in a binary image is its object

contour. The contours of objects are useful in a number of situations such as object analysis (Koplpwitz and Deleone, 1996), pattern recognition (Li et al., 1989), image restoration (Chang and Leu, 1990; Cai, 1988), and the computations of object features (Samet, 1984; Dorst and Smeulders, 1987; Yuan and Suen, 1995) such as the perimeter, area and corner (Chia et al., 2003).

2.2.10.1 Contour tracing

Contour tracing (also called contour following) algorithms trace the boundaries of regions by ordering successive border pixels. In other words, the sequence order among boundary pixels are specified by a contour tracing method based on a counter-clockwise or clockwise traversal (Liow, 1991).

Contour tracing algorithms often consist of three parts (Ren et al., 2002):

1. A starting point is detected for tracing.

2. The next boundary point is found.

3. The condition for terminating the tracing operation is determined.

Boundary tracing can be performed for both binary and grey-scale images. However, only binary images are dealt with in this thesis.

The next paragraph concerns contour representation for three conventional contour tracing algorithms shown in Fig. 2.12 as follows:

The background pixels have the value of 0 and are represented by white colour. The object pixels have the value of 1 and are represented by black colour. The outer (or external) contour has a counter-clockwise direction while the inner (or internal) contour has a clockwise direction. A binary image is scanned from up to down and from left to right (raster scan mode). Tracing is also performed based on 8-connectivity. A contour is an outer contour if it encloses an object region. Similarly, a contour is an inner contour if it encloses a background region (i.e. a hole). Contour points are defined as black pixels belonging to an object whose neighbouring pixels involve at least one background (white) pixel. Meanwhile, the contour line is formed by a set of contour points (Ren et al., 2002; Miyatake et al., 1997).

Three conventional contour tracing algorithms are illustrated in Fig. 2.12. The same example figure has been used so that the results of three algorithms can easily be compared with each other. These algorithms are briefly described in the following (Miyatake et al., 1997):

- **Pixel-centre tracing** (Yokoi et al., 1973; Suzuki et al., 1983): This is the most popular algorithm among the three approaches. Tracing starts by raster scanning of the image to locate the first contour point. Then its 8-connected neighbours are checked in an anti-clockwise direction to find the next border point. This procedure is repeated until when the starting point is revisited

after tracing a closed loop. In order to avoid re-entering the same contour several times as the image is scanned, every contour point once visited is marked. In this way, the loop of contour points constitutes a contour line (Fig. 2.12.a). The contour tracing algorithm continues its raster scanning to find another non-marked starting point each time a transition of pixels from 0 to 1 is detected.

- **Pixel-corner tracing**: In this algorithm, each pixel is regarded as a square. For each contour point, the corners of the square pixel are the subject of tracing. The tracing algorithm starts at a non-marked pixel corner. Then it continues to visit adjacent corners successively. Tracing is terminated when the starting corner is reached again. As a result, the loop is a contour consisting of only horizontal and vertical vector segments as depicted in Fig. 2.12b.

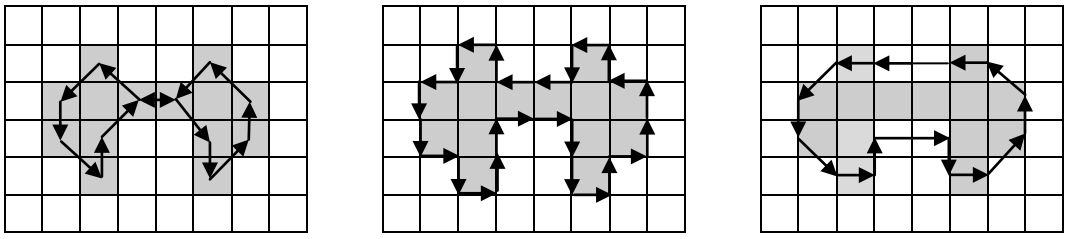
- **Edge-point tracing of run data** (Agrawala and Kulkarni, 1977; Grant and Reid, 1981; Capson 1984): The input to this algorithm is an image expressed by a set of run data. The algorithm checks any two adjacent horizontal scan lines for testing that the runs in both scan lines are connected to each other in a suitable manner. Then the contour is obtained by linking the edge points of the two runs (Fig. 2.12c).

The summary of the features of the above typical algorithms is given in Table 2.2 (Miyatake et al., 1997). Based on this table, it is obvious that each of the mentioned algorithms has its own weaknesses. These drawbacks are either due to tracing speed, accuracy of representation, or required buffer memory size.

Two newer chain code algorithms plus a rapid RLE contour tracing method with better performances are briefly explained as follows:

1. A chain code-based contour tracing algorithm for tracing boundary contours in 2D binary images is presented by Ren et al., (2002). In this algorithm, background and object points are represented by 0 and 1, respectively. It operates as follows (Ren et al., 2002):

    1. A new labelling method is used in this algorithm. The background point neighbouring and out of the outer contour is called OB. The background



(a) (b) (c)

Fig. 2.12 – Contour representation in conventional algorithms: (a) pixel-centre tracing; (b) pixel-corner tracing; (c) edge-point tracing of run data (Miyatake et al., 1997).

	Conventional algorithms		
	Pixel-centre tracing	Pixel-corner tracing	Edge tracing of run data
Tracing speed	Slow	Slow	Fast
Accuracy of restoration	Perfect	Perfect	Imperfect (at concave figure portion)
Accuracy of enlargement	Imperfect (at one-pixel width portion)	Perfect	Imperfect (at concave figure portion)
Required Memory capacity	Large (frame buffer)	Large (frame buffer)	Small (line buffer)
Applications	For small-scale images where tracing is allowed	For small-scale images but high definition needed.	For small-scale images where lower definition tracing is allowed. (e.g. industri al use)

Table 2.2 – Features of three conventional algorithms (Miyatake et al., 1997).

neighbouring and enclosed by the inner contour is called IB. They are both labelled to 2. An object point on a traced contour is labelled to 3. If the change of value of two consecutive points (x, y) and (x+1, y) is from 0 to 1 or from 2 to 1, a new outer contour starting at (x+1, y) is found. However, if the change is from 1 to 0 or from 3 to 0, a new inner contour starting at (x, y) is found. Meanwhile, the labelling operation and contour tracing are performed simultaneously.

2. Once a new contour is found, the tracer is started. Then it examines boundary points one by one according to a predefined sequence which consists of eight 3 x 3 windows (Fig. 2.13). Depending on the current chain code of the tracer (named *pcode*), one of eight 3 x 3 window is selected. Then in that window, relations between the chain codes of points on position labelled 0-6 and *pcode* are orderly examined until the next boundary point is found. In this way, a sequence of contour points is generated.

3. When the tracer reaches the starting point, the closed contour of a region is obtained.

4	3	2
5,a	b	1
x	x	0

(a) *pcode* = 0

4	3	2
5	b	1
6,a	x	0

(b) *pcode* = 1

2	1	0
3	b	x
4	5,a	x

(c) *pcode* = 2

2	1	0
3	b	x
4	5	6,a

(d) *pcode* = 3

0	x	x
1	b	5,a
2	3	4

(e) *pcode* = 4

0	x	6,a
1	b	5
2	3	4

(f) *pcode* = 5

x	5,a	4
x	b	3
0	1	2

(g) *pcode* = 6

6,a	5	4
x	b	3
0	1	2

(h) *pcode* = 7

Fig. 2.13 – The sequence number of examination for next boundary point from the current boundary point *b*.

The algorithm has the following advantages (Ren, et al., 2002):

- Connectivity is preserved and never lost. That is, by utilising this algorithm, the boundary of a region is never split into many segments.

  All region boundaries are traced and no boundary is lost. In other words, the outer and inner contours of all arbitrary complex images are correctly traced. Thus, it is more general and practical.

- The computing complexity of the algorithm is low.

- It is a simple, easily implemented and short time process which traverses the image in one pass.

- By contour filling, which is the reverse operation, a perfect copy of the original image is obtained.

2. Chan and Hsu (2008) describes a shape-preserving contour tracing method for extracting one-pixel-width closed contour for the profile of heterogeneous objects based on grey-scale images. The algorithm called PSCTM (i.e. Pbl Srp Contour Tracing Method) consists of two synch-ronously-performing procedures called PBL (progressive boundary linking) and SRP (synchronous redundancy pruning). The role of BPL is to trace the complete object border pixel-by-pixel using a 3 x 3 sliding window. SRP has the duty of removing random noises and pruning trivial branches/cracks from the contour of the objects simultaneously with PBL.

   The advantages of PSCTM algorithm are (Chan and Hsu, 2008):

   - The extracted contours using PSCTM are qualitatively and quantitatively superior to those extracted using conventional and mathematical morphology methods.

   - In noise-less or non-heavy noisy images, PSCTM has capability to extract one-pixel-width closed contours of the objects in real-time.

   - In comparison to conventional and mathematical morphology cleaning processes, PSCTM preserves the finer details of the object contour and at the same time removes visual redundancies and trivial branches/cracks.

3. A fast finite state machine-based algorithm for the boundary extraction of RLE regions is presented by Quek (2000). It operates directly on the run data structure contained in RLE description of image regions. The boundaries of regions (either external or internal) are obtained in the form of 4- or 8-connected point lists which describe closed positively directed contours of 4- or 8-connected regions, respectively. The algorithm works as follows (Quek, 2000):

   For computing a positively directed region boundary (either external or internal), boundary segments may be grouped into four forms including Up-Right, Down-Left, Down-Right, Up-Left (Fig. 2.14). These segment types constitute the

states of a finite 4-state machine. In fact, the algorithm requires knowing which potions of various runs to be included in the boundary. To do this, it maintains the state of the current traversal as a state in a finite state machine. Transitions between states occur for each move from one segment to the next consecutive segment.

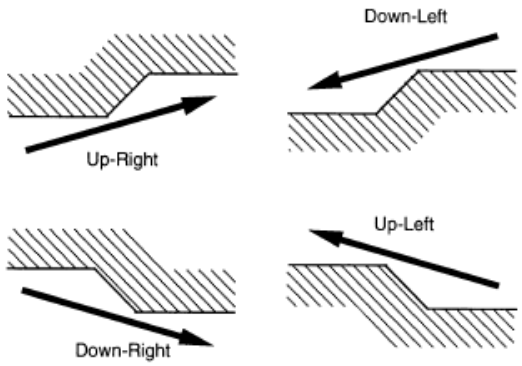


Fig. 2.14 – Boundary segment labels for positively directed boundary (Quek, 2000).

By merging Up-Right and Up-Left into an UP state and Down-Left and Down-Right merging into a DOWN state, the finite 4-state machine may be reduced to two states.

The algorithm tags each run as left-used or right-used. If a run has no tag, it is assumed to be unused (i.e. default case). Two walking functions $WALK_r$ and $WALK_l$ are also utilised which add points to a boundary list moving to the right and left from a starting point toward the specified end of a run. Besides, the top-leftmost unused end of a run is always considered as the starting point of a boundary. If the first unused end encountered is a left-end, the boundary is external and the initial state is set to DOWN. If the first unused end encountered is a right-end, the boundary is internal and the initial state of the machine is set to UP. Once a starting point and the initial state are determined, the finite state machine generates a closed positively directed boundary of a region based on the transition table in Fig. 7.1.

The advantages of this algorithm are (Quek, 2000):

- It is a fast algorithm which operates directly on the run data structure contained in RLE.
- It produces closed positively directed contours for all regions of a binary image.
- Each closed boundary is labelled as internal or external.
- It can handle internal boundaries for each RLE region.
- The algorithm can yield boundaries either in the form of 4- or 8-connected

point lists for describing the contours of 4- or 8-connected regions.

- The utilisation of RLE as an attractive object-based region representation is enhanced by the algorithm as it complements the existing operators which work directly on such region descriptors.

Based on the advantages of the above papers (i.e. Ren et al., (2002), Chan and Hsu (2008), and Quek (2000)), it is concluded that these algorithms are powerful methods for contour tracing of binary/grey scale images. However, as mentioned in sections 2.2.7.1 and 2.2.7.2, both chain code and RLE forms are popular methods for compact representation of foreground regions in binary images. However, some criteria should be determined in order to select one of the above methods for implementation. The selected method should have the following criteria:

- The different processing and computations of foreground regions using the selected method should be simple, efficient, and easily implemented.

- It should be utilised and easily matched with the implementations of a number of blocks of this thesis as illustrated in Fig. 2.1.

One major difficulty with chain codes, as mentioned in section 2.2.7.2, is that they are quite long. So if there are several large regions in each binary image of a video sequence, those regions are represented by long linked lists. Linked list processing is slow especially when the lists are very long. Thus a number of blocks of this thesis may not be performed in an acceptable duration of time if their foreground regions are represented by long linked lists of chain codes. In addition, once foreground regions are represented by their closed contours, for a number of blocks in Fig. 2.1, usually contour information only are not sufficient. Meanwhile, for background subtraction, shadow detection and removal, and object matching and recognition the information of the pixels of foreground regions is also necessary.

Although a linked list is used for each scan line in the run data table of Quek's algorithm, such lists are processed very fast as they are often very short. In addition, Quek's algorithm operates directly on RLE data, used by the representations of other blocks in Fig. 2.1. Therefore, RLE representation has been selected as the more appropriate choice for representing a number of blocks in this thesis including connected components labelling, shadow detection and removal, and boundary extraction and smoothing. For this purpose, Quek's algorithm, as the selected contour tracing method, has been implemented successfully.

2.2.11 Boundary smoothing

Once cast shadow regions are removed, the boundary extraction method can find the silhouettes of all objects. However, the boundaries of objects after segmentation usually have a jagged appearance. Often it is necessary that the outlines of objects are smoothed. Two approaches may be taken:

1. Smoothing the rough outlines of objects can be performed on the binary image.

2. Instead of smoothing objects in the binary image, the boundaries of objects may be transferred to the corresponding input image. Then smoothing can be performed based on the boundaries of objects using colour information.

Consider the first approach. Morphological operators can be utilised for this purpose. Smoothing the outlines of objects in the binary image can be done using the 'closing' (i.e. a dilation followed by an erosion) operation. However, the selection of a structuring element is important. A simple structuring element is needed to only remove single pixel irregularities. For smoothing the outlines of objects usually more than one structuring element is needed. Alternatively, *n* successive dilations followed by the same number of erosions may be applied for smoothing irregularities of *n* pixels in size (Parker, 1997).

Processing of the binary image using morphological operations is fast. The only problem, as already stated, is that morphological operators usually distort the shape of objects. For resolving this problem, a number of contour smoothing methods for binary images are presented in the next sections.

For the second approach, spatio-temporal grey-scale smoothing methods can be applied. Gaussian, mean, and median are samples of smoothing algorithms which can be used. However, Gaussian and mean are linear filters which usually blur edges. Many non-linear algorithms for smoothing have been reported in Literature, too.

Smoothing has extensively been studied in Literature (Xiuwen et al., 2000; Chen, 2000). However, most researchers have paid more attention to grey-scale images than binary ones. In fact, a large number of methods about grey-scale images have been investigated to smooth image noise while important information such as edges is preserved. For this purpose, two main types of smoothing, i.e. linear and nonlinear, are used. Some traditional linear filters are (Lynch et al., 2004; Morse, 2000):

- Averaging multiple frames
- Neighbourhood averaging
- Gaussian smoothing

Averaging multiple frames is a technique to reduce image noise. However, this technique is effective only when a number of frames of the same scene are available. *Neighbourhood averaging* technique replaces a pixel value with the average of the values of the pixels in its neighbourhood (for example, in a window with size n x n, n = 3 or 5). Although only one image is used in this technique, however, it has the drawback that the resulting image is blurred. In *Gaussian smoothing*, each pixel is replaced by a weighted average of its spatial neighbours (Fig. 2.15). The above linear filters have the advantage of reducing the amount of image noise. However, removing or blurring edges are considered as their drawback.

Edge-preserving smoothing algorithms are non-linear filters which have more suitable feature extraction than linear existing smoothing filters (such as Gaussian and mean filters). Some examples of such filters are (Garnica et al., 2000):

- Median Filter
- Symmetrical Nearest Neighbour Filter (SNN)
- Maximum Homogeneity Filter (MHN)
- Conditional Averaging Filter

The features of the above filters are high smoothing degree in homogeneous areas, preserving edges and corners, and conservation of very small homogeneous image regions. Such filters produce filtered grey values without considering the values of neighbouring pixels.

$$1/15 * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Fig. 2.15 – Normalized 3x3 2D Gaussian masks with integer values

The goal of non-linear filters is to remove noise while sharp edges are still maintained. As a result, non-linear filters preserve edges; however, a loss of resolution occurres due to suppressing fine details (Lynch et al., 2004; Morse, 2000).

Due to the complexity of the calculations of edge-preserving algorithms, they need more computation times than simple smoothing methods which are applied to a binary image. On the other hand, for many computer vision applications, which mostly need to work on a large sequence of images in real-time instead of one, speed of computations is a crucial factor. In fact, edge-preserving smoothing methods normally suit image processing tasks operating on a single image but are often not real-time for processing each frame. Thus, in general, non-linear smoothing methods for colour images are very time-consuming processes and may not be appropriate for a sequence of frames.

2.2.11.1 Existing smoothing methods for binary image contours

After the thresholding process, the boundaries of foreground regions often have jagged appearances. The corrupted binary contours are due to discrete sampling (i.e. digitisation), binarisation (i.e. thresholding), and image noise. The sources of noise in the image include the camera, the lenses, the lighting or the signal path. For the following reasons binary image contours should be smoothed (Legault and Suen, 1997; Yu and Yan, 1997):

- Corrupted boundaries make the recognition of foreground regions unreliable.

- A number of measurements such as perimeter, area, moments, tangent slopes, curvature, etc are obtained based on binary contours which are used to represent foreground regions. Thus, no reliable estimates of these measurements can be obtained when binary regions contours are corrupted by noisy pixels.

- It is very important in computer vision and pattern recognition that foreground region contours are represented and processed accurately.

Based on the discussion given in section 2.2.11, it is simpler, much easier and a less time-consuming process if smoothing is performed on binary contours. Thus, some boundary smoothing algorithms should be sought which only operate on the boundary pixels effectively. In this regard, a review of the smoothing methods for binary contours is given in the next sections.

2.2.11.2 Binary contour smoothing using chain codes

Several smoothing methods for binary contours based on chain codes have been proposed in the literature as follows:

- Suen et al. (1992) applied a simple contour smoothing technique. For each contour point, its coordinates are replaced by averaging them with the coordinates of the neighbouring points (i.e. the preceding and the following points). This technique is applied twice around the entire contour.

- Legault and Suen (1997) proposed an algorithm for obtaining optimal local weighted methods defined as Eq. 8.1:

$$\mathbf{P}_i^{(k)} = \sum_{j=-n}^{n} \alpha_j \mathbf{P}_{i+j}^{(k-1)} \quad , k = 1, 2, ..., k \qquad \text{(Eq. 2.7)}$$

  The constant coefficients α_j should be determined for smoothing binary contours according to specific goals such as accurate estimation of point positions, slopes of tangents, or deviation angles from point to point. For this purpose, a simple model based on an infinite horizontal border with random one-pixel noise is considered. The goal of this algorithm is to eliminate the "wiggles" along the noisy horizontal border as much as possible so that after k smoothing iterations, the border pixels are "as straight as possible" (Legault and Suen, 1997). The major drawbacks of this algorithm are its high computational load due to k smoothing iterations which require floating point operations and removing just one-pixel noise.

- Yu and Yan (1997) proposed an effective sequential algorithm for binary contour smoothing using difference chain codes. The algorithm is able to remove noisy pixels along a contour and convert boundary points to a set of straight lines. It also finds convex and concave segments along the contour based on detecting structural feature points. The algorithm has the advantage of operating only on contour chain codes without requiring any derivative-based corner detection procedure. So it is very fast and its implementation is efficient

(Yu and Yan, 1997). The weakness of the algorithm is considered as just one boundary point smoothing.

- Hu et al. (1998) proposed a similar algorithm for binary contour smoothing based on the work of Yu and Yan (1997). The new method is called "multiple-point smoothing algorithm" for further smoothing of the boundary. After the algorithm's operation, the noise in boundary contours is greatly reduced and the chain code properties are preserved. Also three kinds of feature vectors (even and odd chain code directions and bending points) are extracted for each boundary. The advantages of this algorithm are:

  - The feature vectors have more effectivity in comparison with the previously proposed feature points.

  - The algorithm's operations including the smoothing and feature extraction are very fast because they are based on chain codes and use no float-ing point operations.

2.2.11.3 Binary contour smoothing using RLE representation

Run-length smoothing algorithm (RLSA) was introduced by Wong et al. (1982) and taken up again by Wang and Srihari (1989). It is a low complexity technique and is often used for segmenting printed documents such as newspapers into rectangular blocks. Then the algorithm classifies these blocks into meaningful regions, for example, text, graphics, and halftone image regions. It is common that documents are printed as dark points (represented by 1's) on a light background (represented by 0's). The main idea of RLSA is to eliminate white (background) runs in the horizontal and vertical directions and replace them with black (foreground) runs. For this purpose, any two black (foreground) pixels (1's), which their distances are less than or equal than a threshold *sv*, are merged into a stream of dark pixels. White (background) pixels remain unchanged. For example, with *sv* = 3 and the input sequence: 000110000001100100001, the result of RLSA on this sequence will be: 111110000001111100001 (Wang and Srihari, 1989).

RLSA usually consists of three main stages. First, in the binary image, RLSA is applied row-by-row to eliminate horizontal runs whose lengths are smaller than a threshold *hsv*. Then RLSA is applied column-by-column to eliminate vertical runs whose lengths are smaller than a threshold *vsv*. The resulting two bit maps are subsequently combined with each other by a logical *AND* operation to produce a new smoothed image. However, there are some small gaps among the blocks of text lines in this image. An additional horizontal smoothing operation is performed by using a new threshold value *ahsv* to produce the final smoothed image. RLSA has an effect of a smear on the binary image by connecting together black pixels of the image which are closely located. Once the major blocks of a document are determined, the next stage extracts the text lines (Papamarkos et al., 1996; Wang and Srihari, 1989).

The main disadvantages of RLSA are as follows (Papamarkos et al., 1996):

- RLSA uses two/three parameters *hsv* and *vsv* (and sometimes *ahsv*) threshold values for horizontal and vertical directions. These two/three parameters should be determined manually based on some heuristic methods.

- The method is not robust because if the assumptions made for the determination of the heuristic parameters are not satisfied, the method will fail.

Papamarkos et al. (1996) proposed an algorithm for automated calculation of the proper horizontal and vertical smoothing values (i.e. *hsv* and *vsv*) of RLSA method. These parameters are calculated based on the mean character length (*mcl*) and the mean text line distance (*mltd*) of the document. The contributions of the horizontal and vertical run-lengths determine the values of *mcl* and *mltd* (Papamarkos et al., 1996).

As discussed in section 2.2.10.1, RLE representation has been chosen for connected components labelling, shadow detection and removal, and boundary extraction and smoothing. In addition, Quek's RLE contour tracing method was implemented to extract the boundaries of foreground regions in each binary image. Meanwhile, as stated above, RLSA is more suitable for document smmothing but it is not approriate for smoothing the boundaries of foreground objects in an image sequence. Fortunately, a simple and fast smoothing method for smoothing the jagged outlines of binary objects will be proposed in ***Chapter 7*** (for block number 10 in Fig. 2.1). Other advantages of the proposed method are smoothing only the boundaries of objects while preserving their shapes with no or very little distortion.

While smoothing, the RLE-based data structures, which store information of the objects, should be corrected based on changes created in the outlines of the objects. Thus, after smoothing, the RLE data structures will also be updated to indicate all changes created on the boundaries of objects.

2.2.12 The remaining blocks

For the next stages, including object recognition and tracking (block number 11) and object behaviour understanding (block number 12), required information is available. In fact, after low and mid-level processing, each object has its own feature profile. For each object, a separate feature profile is stored in the data structure(s) of the computer vision system which may contain the following items:

1. The start and end addresses of each line of pixels of a region corresponding to an object, stored in a RLE-based data structure.

2. The boundary pixels of each object.

3. If an object has a cast shadow, its shadow region information should be stored in a separate RLE-based data structure. The outline pixels of the shadow region should also be stored.

From the above information all other information can be obtained and stored in

object feature profile. It may include the following items:

- Geometrical features such as perimeter, area, object centre, medial axes, etc.
- Colour or texture features such as average colour, necessary information of the homogeneous segments within the object regions, etc.
- Temporal features such as speed, acceleration, distance from a specified origin, motion direction, motion trajectory, etc all computed for each frame and in comparison to the image plane.
- Spatial features similar to all temporal information given above but in the frequency domain.
- All above information concerning the cast shadow of each object (if there is any cast shadow).

Thus, depending on a specific application, a computer vision designer can decide about the required features which the vision system should keep for each object.

From the above discussion, it is obvious that the effectivity of high-level processes like recognition, tracking, interpretation and understanding depends on previous stages. So for successful operations of high-level processes, low- and mid-level processes must do their best to produce the most accurate and correct information. Any minor failure of low- and mid-level processes may cause the later processes to become ambiguous or fail. In the worst case, an autonomous computer vision system may crash by providing erroneous results. Therefore, it can be concluded that the more accurate the results of low- and mid-level operations are, the more effective and robust the high-level processes will be.

Chapter 3

Reviewing Background Removal Algorithms

3.1 Introduction

In this chapter, background removal algorithms will be reviewed. First, the problem and the main goal of all background removal methods are stated in Section 3.2. Then in Section 3.3, a summary of different approaches in the literature are given. Finally in Section 3.4, a brief comparison of salient features of the most important methods is presented.

3.2 Problem statement and requirements for background removal methods

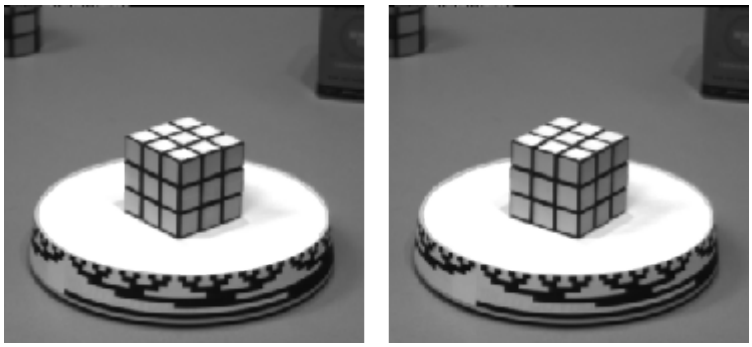
In numerous computer vision applications, extracting moving objects from a video sequence captured using a static camera is a very important task. Some typical applications are video surveillance, traffic monitoring and analysis, human motion detection and tracking, industrial automation, etc. The main goal in the above areas is to “detect all foreground objects”.

Three common approaches to moving target detection are *optical flow* (Fejes and Davis, 1998; Wixen and Hansen, 1999), *temporal differencing* (Anderson et al., 1985) and *background subtraction* (e.g. Haritaoglu et al., 2000; Toyama et al., 1999; Wren et al., 1997) as explained in the following subsections.

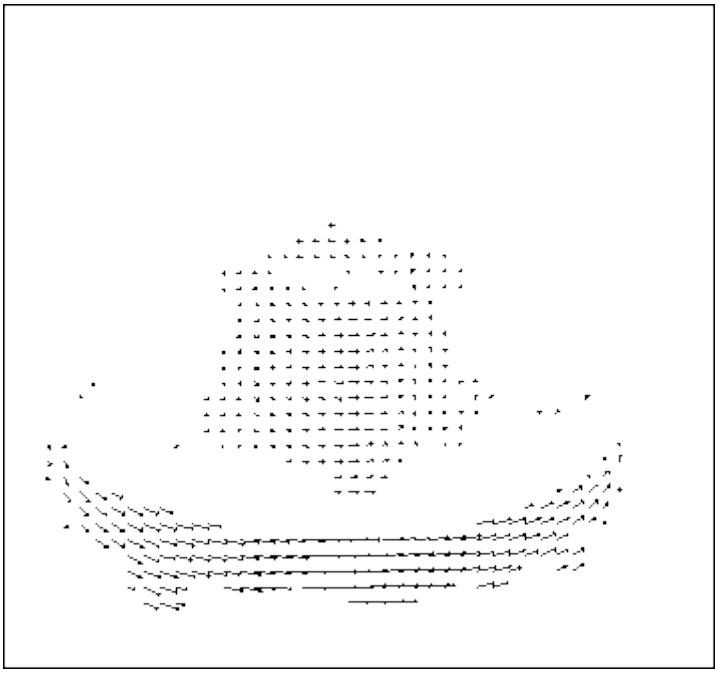
3.2.1 Optical flow

If an observer (a camera or a human eye) moves in a 3D scene, the pattern of motion of objects, surfaces, and edges in a visual scene caused by the relative motion between the observer and the scene is called *optical flow*. The direction and the speed

of motion of the features in the visual scene can be described by optical flow. The magnitude and direction of optical flow at each position is represented by the direction and the length of an arrow. In an image sequence, motion can be estimated as either instantaneous velocities or discrete image displacements. Fig. 3.1b shows the optical flow pattern from two images of a rotating Rubik's cube shown in Fig. 3.1a.



(a)



(b)

Fig. 3.1 – (a) A Rubik's cube on a rotating turntable; (b) Flow vectors calculated from comparing the two images of a Rubik's cube (Russell and Norvig, 1995)

Differential optical flow methods calculate the motion between two image frames taken at times t and $t + \Delta t$ at every pixel position based on local Taylor series approximations of the image signal. These methods utilise partial derivatives with respect to the spatial and temporal coordinates.

Motion estimation, video compression, object detection and tracking, movement

detection, robot navigation and visual odometry are a number of optical flow applications. As an example, target detection and tracking can be performed using optical flow even though there is no prior knowledge about the background or when the camera is moving. But optical flow methods need very complex computations which cannot be coded in real-time algorithms without specialised hardware (Spagnolo, et al., 2003). In addition, optical flow methods are also sensitive to noise. Thus, many researchers usually do not prefer using optical flow methods for implementing real-time background generation techniques.

3.2.2 Temporal differencing

Temporal differencing (or frame differencing (Hou and Han, 2004)) is a technique by which the arithmetic difference of corresponding pixels in the same physical locations in two frames of an image sequence is obtained. The first variant of this technique uses two consecutive frames (Jain, 1981). The difference image contains non-zero values whenever objects have moved to another location but will be black when no moving object is detected. The problems with the earlier variant are as follows (Yang and Levine, 1992):

1. A pre-selected threshold is necessary to obtain the thresholded difference image. As a result, this method will be dependent on the video sequence and selected threshold.

2. If an object moves too slowly so that it is stationary in two consecutive frames, the object will not appear in the difference frame.

3. Extending the method to use more than two frames is difficult.

In later variants, the extracted frames from the sequence are at time t_0 and time t_k (the selected frames are not necessarily consecutive). The advantages of these variants are that they can detect targets in real-time and are adaptive for dynamic environments (Neri et al., 1998; Thomas and Ngan, 1998). However, they have the following disadvantages (Spagnolo, et al., 2003):

1. Generally, they poorly extract the entire relevant feature pixels and the real shapes of objects are not obtained. They also do not produce closed object contours. Therefore, morphological operations (e.g. dilation) or an edge closure algorithm e.g. guided by the edge gradient must be added (Cucchiara, et al. 2000). Meanwhile, the interior regions of objects are detected as static. Thus, possibly holes inside moving objects are generated.

2. The conventional method relies on incorporating only two consecutive frames. Improved variants of this method are also presented (Collins, et al., 2000, Jolly, et al., 1996, Yoshinari and Michihito, 1996), which use three-frame subtraction (i.e. double frame difference) to extract the moving objects. Double frame difference is more precise in locating real objects than single frame difference. Although better results are obtained using these improvements, usually they still have shortcomings of temporal differencing algorithm.

Temporal differencing (single or double frame difference) techniques can be useful when the object's motion is mainly along a known direction, for instance when a road is monitored with coming and going vehicles (Cucchiara, et al. 2000).

3.2.3 Background subtraction

The most important and most common approach to identify and segment moving objects in a video stream is background subtraction, which involves computing a reference image for each new frame. Then, by comparing the next input frame with the reference image, regions of the image which have changed are identified. Thresholding the result produces a binary segmentation which is used for discriminating the moving objects from the background regions.

In the simplest case, background subtraction can be performed by subtracting each incoming frame from a reference image. As reference image an initial input frame may be selected, which does not include any moving object. However, sometimes it is not practical to select an initial input frame as a reference image. Also in some situations, there is no possibility to obtain an initial model using a short training sequence which contains no moving object. For example, it may be impossible for a traffic surveillance system which monitors a busy street or a highway to acquire a background image with no moving object. As another example, it may be difficult or impossible in some situations, such as public areas, to control the area being monitored, which are characterised by a continuous of moving objects or other disturbing effects (Gutchess, et al., 2001).

As already indicated in Section 2.2.3, due to several factors, the background image must be temporally adaptive and it should be updated continuously in order to be kept up to date. Some of these factors (i.e. background generation requirements) are (Piccardi, 2004):

- Illumination changes
  - Gradual variations in lighting conditions.
  - Sudden illumination changes (such as turning a lamp on or off in an indoor environment or abrupt changes of light level in an outdoor environment due to covering or uncovering the sun by clouds).
- Motion changes
  - Camera oscillations.
  - Small movements of background objects such as trees waving in the wind or sea waves, rain, snow, etc.
- Changes in the background geometry

- Introducing or removing objects in the scene (such as a door is opened and then is left opened or a parked car is moved on).

➢ An initialisation process is required by a number of algorithms.

These problems and requirements are the constraints which should be considered as important characteristics by adaptive background removal algorithms.

There are a lot of background removal algorithms in the literature which the flow diagram of most of them consists of four major steps as follows (Cheung and Kamath, 2004; Elhabian et al., 2008):

1. Pre-processing (e.g. including simple tasks such as converting the raw input video into a format suitable for later processing steps)

2. Background modelling (also called background maintenance)

3. Foreground detection (also known as background subtraction)

4. Data validation or post-processing (for removing pixels that do not belong to actual moving objects)

In the following these steps are explained (Elhabian et al., 2008):

1. Pre-processing steps are performed to filter out unimportant changes or to do some initial actions before any decision about object detection is made. These steps include:

   ➢ Geometric adjustments: Camera motion may cause intensity changes of pixels which are never desired to be detected as real changes. Thus, frame registration is utilised to align several frames into the same coordinate frame.

   ➢ Radiometric/Intensity adjustments: Pre-compensation for illumination variations between frames may be performed which is due to changes the strength or the position of light sources in the scene. For example, the pixel intensity values are normalised both for the current input frame and its corresponding background image to have zero mean and unit variance.

   ➢ Image derivatives: Sometimes pixel-based features such as spatial and temporal derivatives are utilised by a number of algorithms to incorporate edges, level lines, and motion information, which are invariant to illumination changes, to have a suitable representation of the scene background.

   ➢ Data reduction: In order to reduce the data processing rate, frame-size and frame-rate reduction are commonly used by real-time systems.

   ➢ Noise reduction: As an initial pre-processing stage, simple temporal and/or spatial smoothing is often used to reduce camera noise and to remove

transient environmental noise such as snow and rain captured in outdoor scenes.

- Format transformation: Pre-processing may include feature extraction such as transforming input frames into the most appropriate feature space (e.g. converting data format) derived only from input frames. Colour images have become popular for background removal algorithms. The RGB space is generally used colour space since RGB values are readily provided by most frame grabbers. However, RGB colour space does not behave well with respect to colour perception because a distance computed between two colours in RGB space does not show their perceptual similarity. Thus, some algorithms transform RGB colour space to other spaces such as YUV, HSV, normalised rgb, C1C2C3, etc as a pre-processing step.

2. Background modelling: The heart of any background removal algorithm is background modelling, also known as background maintenance. According to Cristani et al. (2003), a background modelling process is usually characterised by three issues; model representation, model initialisation, and model adaptation. The kind of model used to represent the background is described by the first one. The second one is concerned with how the model is initialised and the third one regards the mechanism of adapting the model to background changes (e.g. illumination changes).

3. Foreground detection: For detecting foreground objects, each input frame is compared with the background model to identify candidate foreground pixels in the input frame. The result of this comparison is called difference image or difference map, which is binarised by thresholding to classify foreground pixels. The correct threshold value is determined based on the scene, the camera noise, and the illumination conditions.

   There are a number of methods to generate the difference map including absolute difference edge-based, relative difference, predictive-based, statistic-cal-based approaches such as single Gaussian-based, mixture of Gaussians-based, kernel density estimation-based, etc. However, one of the simplest but effective methods is absolute differencing, which was explained in section 2.2.4.

4. Data validation: The output of a foreground detection algorithm is a binary frame called foreground mask. It generally has a noisy appearance with isolate-ed foreground pixels and a number of connected foreground components with jagged silhouettes and holes inside them. The process of improving the foreground mask based on the information outside of the background model is known as data validation, which is sometimes referred to as the post-processing phase (Cheung and kamath, 2004).

   False positive and false negative misclassifications may occur in the segmentation of the foreground mask. Data validation aims to reduce the number of false positives (i.e. background regions incorrectly classified as

foreground) and false negatives (i.e. foreground regions mistakenly labelled as background) without an essential degradation in classification speed.

Simple standard binary image processing operations such as using median fitters to remove small groups of pixels or morphological operations to smooth object boundaries may be used as the post-processing phase by some algorithms. However, more sophisticated post-processing operations can be applied to both the foreground mask and the original image. For example, typical features such as motion, colour and edge information can be analysed to improve the spatial accuracy of detected foreground regions.

A brief review of the most important background removal algorithms will be given in the next section. The goal is to observe how much the vision researchers have been successful to satisfy the constraints mentioned in this section.

3.3 A review of background removal algorithms

Background modelling is the core of all background removal algorithms. Many researchers have developed a background model that is robust against environmental changes in the background. The goal of using background removal methods is to detect all moving objects of interest. Background removal methods can be classified into two broad categories: non-recursive and recursive (Cheung and Kamath, 2004). These two categories are described in the following subsections.

3.3.1 Non-recursive techniques

A non-recursive technique uses a sliding-window approach containing a number of last frames for background estimation. They maintain a buffer to hold N previous input frames. The background image is estimated based on the temporal variations of each pixel in the buffered frames. Only the frames within the buffer (inside the sliding-window) are used for estimating the background and more previous input frames have no effect on the computations. A significant storage requirement is considered as the weakness of these techniques as a large buffer may be needed by some methods to cope with slow-moving objects. One solution to alleviate the problem of keeping a large number of frames is to store video sequences at a lower frame rate r (Cheung and Kamath, 2004). A number of commonly-used non-recursive techniques are as follows:

- **Frame differencing:**

  One of the oldest methods of background generation is the frame difference algorithm (Jain, 1981). In this method, which uses the difference of two consecutive frames, the previous input frame is considered as the estimated background. Obviously this algorithm can only work in special conditions of objects' speed and frame rate (Piccardi, 2004). The advantages and drawbacks of this method have also been mentioned in Section 3.2.2.

- **Mean filter:**

  A simple approach for estimating the background image can be the average over the last *N* input frames containing no moving object. However, in situations where foreground objects are inevitably visible in the video stream (such as in public areas) the mean cannot produce a correct background image. This happens when the number of frames *N* is small and the difference in luminance between background and the moving objects is high. In such cases, especially when there are lingering or slow-moving objects in a video sequence, a blurring effect of those objects occurs which makes the background image unacceptable.

- **Mode filter:**

  Some articles propose to perform the background update using statistical functions on a sequence of the most recent samples such as *mode* function (Shio and Sklansky, 1991). They argue that the most correct estimation for the background is given by the *mode* which gives the most probable luminance value as the maximum of the probability distribution. However, for implementing the *mode* function, enormous data structures (i.e. a linked list with *N* nodes for each image pixel) need to be kept in the memory (Cucchiara, et al. 2000).

- **Median filter:**

  Temporal median filtering is another statistical function, which is utilised as one of the most commonly used background modelling techniques. For this filter, the assumption is that pixels remain in the background for more than half of the frames in the buffer. For example in Rosin and Ellis (Rosin and Ellis, 1995), each pixel in the reference image is obtained as the result of applying a median filter to the input image at the same pixel ($B_{x,y} = \text{median}_t \, I^t_{x,y}$).

  In contrast to the mean function, which is very fast, the median function requires sorting the last *N* samples of each image pixel. The complexity of computing the median is O (*N* log *N*) for each pixel. In addition, temporal median filters work well if objects have some movement. Thus, if foreground objects have few movements, gradually some parts of objects are transferred to the background and the difference image will be zero for those regions.

The mean, mode and median filters are called *basic* methods. In these three methods, the background model at each pixel location is updated using the recent history of the pixel (e.g. *N* previous frames). Moreover, no spatial correlation between the locations of neighbouring pixels is used (Piccardi, 2004).

- **Selectivity:**

  Methods, which exclude moving object pixels from background update, are called *selective background update* algorithms (Cucchiara, et al., 2000;

Elgammal, et al., 1999). However, using selectivity causes a problem concerning ghosts; if ghosts are excluded from the background update, the background will never be correctly estimated, and the ghosts will be permanently deleted (Cucchiara, et al., 2003). Thus, adaptive background generation algorithms should precisely consider the ghost problem.

The basic methods can apply selectivity in their approaches to have a better performance. The utilised procedure is as follows (Piccardi, 2004):

- Each pixel in every input frame is classified as either a foreground or a background pixel.

- If the pixel is detected as a foreground point, it is ignored in the background update process.

As a result, the pixels, which logically do not belong to the background scene, are prevented from the background update process.

- **Least median of squares:**

Yang and Levine (1992) proposed an algorithm for constructing the background primal sketch, which is an edge map of the background without moving objects. They built the background primal sketch by taking the median value of the pixel colour over a series of images. Their suggestion was to compute B_t (x, y) using the *least median of squares* (LMedS) estimate, i.e.

$$B_t(x, y) = \min_b \text{median}_t (I_t(x, y) - b)^2 \qquad \text{(Eq. 3.1)}$$

The disadvantages of this method are the need for various parameters as well as the requirement of a continuously unoccluded view of the background (Rosin and Ellis, 1995).

- **Linear predictive filter:**

The *Wallflower* method (Toyama, et al., 1999) uses a three tiered algorithm for the background subtraction problem. The background is modelled by (Javed, et al., 2002):

1. Pixel-by-pixel linear prediction using colour information

2. Region-level by region filling algorithm for dealing with background object relocation problem

3. Frame-level by model switching for detecting global illumination changes

The current background estimate is computed by applying a simpler version of the Kalman filter called *Weiner* filter on the pixels in the buffer (Mittal and Paragios, 2004). This algorithm can only handle sudden changes in

illumination if the model describing the scene after the illumination changes is known a priori (Javed, et al., 2002).

The *Wallflower* method assumes that an initial model can be obtained using a short training sequence which contains no foreground object. However, it is sometimes difficult or impossible to control public areas. So the initialisation process may malfunction in such cases. Moreover, the filter coefficients are estimated at each frame based on the sample covariances, making the technique difficult to apply in real-time (Cheung and Kamath, 2004).

- **Non-parametric background model:**

In the non-parametric background model (Elgammal, et al. 1999) or kernel density estimation (KDE), the background probability density function (*p.d.f*) is obtained by the histogram of the *N* most recent pixel values as follows:

$$f(I_t(x, y)) = \frac{1}{N} \sum_{i=t-N}^{t-1} K(I_t(x, y) - I_i(x, y)) \qquad \text{(Eq. 3.2)}$$

where $K(.)$ is a smoothing Gaussian kernel estimator. The pixel I_t (x, y) is considered as a foreground point if there is little possibility that f (I_t (x, y)) has such a distribution, i.e., f (I_t (x, y)) is smaller than one global threshold value τ. Utilising the full density function for a single estimation has the advantage of representing multimodal background such as waving trees, ocean waves, rain, snow, moving clouds, etc. Moreover, the algorithm has the advantage of increased detection sensitivity and simultaneously reduces false positives (Cheung and Kamath, 2004).

The non-parametric kernel density algorithm uses the median of the absolute differences between successive frames as the width of the kernel. Thus, it requires intensive computations for kernel values of each point in order to declare it as a foreground pixel. In addition, this method needs a significant memory requirement (*N* * size (frame)).

- **Standard mean-shift based estimation:**

In this gradient-ascent method, the modes of a multimodal distribution are detected using their covariance matrix. The method uses an iterative approach so its step is decreased until it is converged. For *n* data points $\mathbf{x_i}$, $i = 1 \ldots n$ in the *d*-dimensional space *Rd*, the multivariate mean shift vector computed with kernel *g* in the point **x** is given by:

$$m(x) = \frac{\sum_{i=1}^{n} x_i \, g((x - x_i/h)^2)}{\sum_{i=1}^{n} g((x - x_i/h)^2)} - x \qquad \text{(Eq. 3.3)}$$

where *h* is the kernel bandwidth (Comaniciu, 2002).

The major problems with the standard mean-shift method are that the algorithm is very slow and requires the amount of "*N* (i.e. buffer length) * size (frame)" memory (Piccardi, 2004).

- **Eigenbackground subtraction:**

The eigenspace model is formed by taking a sample of *N* frames and these frames are re-arranged as the columns of a matrix A. Then, both the mean μ (background image) and its covariance matrix $C = AA^T$ are computed. By an eigenvalue decomposition, the covariance matrix can be diagonalised as $L = \Phi C \Phi^T$, where Φ is the eigenvector matrix of the covariance of the data and L is the corresponding diagonal matrix of its eigenvalues. Only the first *M* eigenvectors (eigenbackgrounds), corresponding to the *M* largest eigenvalues, are kept to give a Φ_M matrix. A principle component feature vector $I_i - \Phi^T_M X_i$ is then formed, where $X_i = I_i - \mu$ is the mean normalised image vector. Each input image I_i is projected into an *M* eigenvector subspace, i.e. $B_i = \Phi_M X_i$ to model the static parts of the scene, pertaining to the background. By thresholding the Euclidean distance between the input image and the projected image, i.e. $D_i = | I_i - B_i | > \tau$, where τ is a given threshold, the moving objects are detected (Oliver, et al., 2000).

The authors of the paper (Oliver, et al., 2000) state that it works well and is faster than the MOG (mixture of Gaussians) approach (Piccardi, 2004).

3.3.2 Recursive techniques

Recursive techniques do not use buffer for background estimation. Instead, a single background model is updated using each input frame recursively. Often more weights are given to most recent samples and as a result, input frames from past distance usually have less effect on the current background model (Cheung and Kamath, 2004).

In comparison with non-recursive techniques, recursive techniques require much less storage. However, if an error object suddenly appears in the background image, it may remain in the background for a much longer period of time. Some recursive techniques are described as follows:

- **Running average:**

The background image is updated for comparison with the next input frame as follows:

$$B_{t+1}(x, y) = \alpha * I_t(x, y) + (1 - \alpha) * B_t(x, y) \qquad \text{(Eq. 3.4)}$$

where $0 < \alpha < 1$. α is called the learning rate and is typically 0.05. If this updating process is used, only the current and the previous background image should be stored.

- **Running average with selectivity:**

  In this approach, the background model uses both running average and selectivity for updating background pixels as follows (Piccardi, 2004):

  $B_{t+1}(x, y) = \alpha * I_t(x, y) + (1 - \alpha) * B_t(x, y)$: if $I_t(x, y)$ is a background pixel (Eq. 3.5)

  $B_{t+1}(x, y) = B_t(x, y)$: if $I_t(x, y)$ is a foreground pixel (Eq. 3.6)

- **Approximated median filter:**

  In this technique, the running estimate of the median is incremented by one if the input pixel is larger than the estimate, and decreased by one if smaller (McFarlane and Schofield, 1995). This estimate eventually converges to a value for which half of the input pixels are larger than and half are smaller than this value, that is, the median (Cheung and Kamath, 2004).

- **Kalman filter:**

  Kalman filter is a widely-used recursive technique for tracking linear dynamical systems under Gaussian noise (Cheung and Kamath, 2004). Different versions of Kalman filtering have been proposed for background modelling. In the simplest version, only luminance intensity is used (Halevy and Weinshall, 1999; Boult et al. 1999; Wren et al. 1997). Karman and von Brandt (Karman and von Brandt, 1990) apply intensity and its temporal derivative. Their approach is able to adapt with lighting and weather temporal changes. However, there are two difficulties in this method. Firstly, the method needs parameters which should be set manually. Secondly, inappropriate selection values for two parameters of the method leads to severe performance degradation. Thus, this technique cannot be used in systems which need to produce background image automatically.

  Ridder et al. (1995) modelled each pixel with a Kalman filter which made their system more robust to lighting changes in the scene. While this method uses a pixel-wise automatic threshold, it still recovers slowly and does not handle bimodal backgrounds well (Stauffer and Grimson, 1999).

- **Running Gaussian average:**

  Pfinder (Wren et al., 1997) uses colour images and a statistical model of the background instead of a reference image. It assumes that most of the time, the system processes a scene containing a relatively static situation (i.e. a slow-changing environment). For this purpose, this approach models the background as a textured surface, each point of which is associated with a mean colour and a variance about the mean. Thus, Pfinder fits one Gaussian distribution (μ, σ) over the histogram which gives the probability density function (*p.d.f*) of the background. It also applies the running average to update the background *p.d.f* as follows:

$$\mu_{t+1}(x, y) = \alpha * I_t(x, y) + (1 - \alpha) * \mu_t(x, y) \quad \text{(Eq. 3.7)}$$

$$\sigma^2_{t+1}(x, y) = \alpha * (I_t(x, y) - \mu_t(x, y))^2 + (1 - \alpha) * \sigma^2_t(x, y) \quad \text{(Eq. 3.8)}$$

The method uses a threshold for partitioning the background pixels into visible and occluded points by examining the following test:

$$| I_t(x, y) - \mu_t(x, y) | > \tau, \qquad \text{where } \tau = k\sigma. \quad \text{(Eq. 3.9)}$$

In each frame, the statistics of visible pixels are updated using a simple adaptive filter in order to compensate for background changes due to illumination and human motion (Spagnolo, et al., 2003). However, for Pfinder to operate correctly in indoor environments, the room must be kept empty during the initialisation period. Thus, this system is sensitive to initialisation inaccuracies. In addition, Pfinder has another difficulty in modelling the background in outdoor scenes because it cannot handle small motions of background objects such as waving trees. Therefore, it cannot cope with multimodal backgrounds (Piccardi, 2004).

- **Mixture of Gaussians:**

In the mixture of Gaussians (MOG) model (also called Gaussian mixture model or GMM), the intensity of each background pixel is adaptively represented by the summation of *k* weighted Gaussians (Stauffer and Grimson, 2000 and 1999). The MOG model maintains a density function for each pixel and as a result is capable of handling multimodal background distributions. The number of modes (i.e. *k*) is usually predefined from 3 to 5. The pixel distribution f (I_t (x, y) = *u*) is modelled as a mixture of *k* Gaussians (Cheung and Kamath, 2004):

$$f\ (I_t(x, y) = u) = \sum_{i=1}^{k} \omega_{i,t} \ . \ \eta\ (u;\ \mu_{i,t},\ \sigma_{i,t}) \quad \text{(Eq. 3.10)}$$

where $\eta\ (u;\ \mu_{i,t},\ \sigma_{i,t})$ is the *i*-th Gaussian component with intensity mean $\mu_{i,t}$ and standard deviation $\sigma_{i,t}$. $\omega_{i,t}$ is the portion of the data accounted for by the *i*-th component. For each input pixel I_t (x, y), the component $\hat{i}$ whose mean is closest to I_t (x, y) is declared as the matched component if

$$| I_t(x, y) - \mu_{\hat{i},t-1} | \leq 2.5\sigma_{\hat{i},t-1} \quad \text{(Eq. 3.11)}$$

At every new frame, the parameters of the matched component are then updated as follows:

$$\omega_{\hat{i},t} = (1 - \alpha)\ \omega_{\hat{i},t-1} + \alpha \quad \text{(Eq. 3.12)}$$

$$\mu_{\hat{i},t}(x, y) = (1 - \rho)\ \mu_{\hat{i},t-1}(x, y) + \rho I_t(x, y) \quad \text{(Eq. 3.13)}$$

$$\sigma^2_{\hat{i},t}(x, y) = (1 - \rho)\,\sigma^2_{\hat{i},t-1}(x, y) + \rho(I_t(x, y) - \mu_{\hat{i},t}(x, y))^2 \quad \text{(Eq. 3.14)}$$

where α is a user-defined learning rate with $0 \leq \alpha \leq 1$. ρ is the learning rate for the parameters and can be approximated by

$$\rho \approx \frac{\alpha}{\omega_{\hat{i},t}} \quad \text{(Eq. 3.15)}$$

Actually the mixture of Gaussians models both foreground and background. In order to determine whether $I_t(x, y)$ is a foreground or background pixel, all components are ranked by their $\omega_{i,t} / \sigma_{i,t}$. If i_1, i_2, …., i_k is the component order after sorting, the first M components that satisfy the following criterion are declared to be the background components (Cheung and Kamath, 2004):

$$\sum_{k=i1}^{iM} \omega_{k,t} \geq \Gamma \quad \text{(Eq. 3.16)}$$

where Γ is the weight threshold.

The advantages of Gaussian mixture models (GMMs) are dealing with lighting changes, slow-moving objects, and introducing or removing objects from the scene. A drawback of GMM algorithms is that the number of the mixture components is pre-set and fixed-value. GMM approach for foreground segmentation is a time-consuming process. This is due to estimating the number of parameters which are mostly determined by the number of mixture components. Also the application of GMMs for background subtractiton requires an efficient method for learning the GMM parameters which are computationally expensive. Therefore, the selection of the number of components and the initialisation process are two important problems of the GMM algorithm for background subtraction (Cheng et al., 2006).

GMM algorithms have other disadvantages. Using a few Gaussians may not easily model the fast variations of the background. Thus it may be unable to provide sensitive detection (Elgammal, et al., 1999). Depending on the learning rate, GMM algorithms may also encounter trade-off problems for adapting to background variations. They may fail to detect sudden illumination changes in the background due to a low learning rate which produces a wide model. The background model absorbs slowly moving foreground pixels once the model adapts too quickly. As a result, this phenomenon, called the foreground aperture problem (Toyama et al., 1999), produces a high false negative rate (Kim et al., 2005). Nevertheless, due to the popularity of this method, a number of techniques have been developed to improve the performance of the GMM algorithm (Tian, et al., 2005, Eng et al., 2004, Harville, 2002, Javed, et al. 2002). For example, Figueiredo and Jain (2002) proposed an unsupervised algorithm for learning a finite mixture model which has two properties (1) it selects the number of components auto-

matically (2) it is less sensitive to initialisation (Cheng et al., 2006). Zivkovic and van der Heijden (2006) developed Figueiredo and Jain's research (2002) and proposed an online algorithm which automatically selects the required number of components per pixel. The advantage of the new method is that it can fully adapt to the observed scene. Its other advantages are reduced processing time and improved segmentation result (Zivkovic and van der Heijden, 2006).

- **Sequential kernel density approximation (SKDA):**

  In this method, the density is represented by a weighted sum of Gaussians, whose number, weights, means and covariances are updated at each time step to include the new data into the model (Han, et al., 2004).

  The approach uses a variable-bandwidth mean shift mode which detects samples just at initialisation time. For each mode, a Gaussian component is created whose mean is given by the mode location. The covariance of the Gaussian is also derived from the Hessian matrix which is computed at the mode location.

  This method relies on the modelling and density modes which are propagated by adapting them with the new samples as follows:

  $$\text{Probability density function } (x) = \alpha\,(new\_mode) + (1-\alpha)\,(\sum existing\_modes) \quad \text{(Eq. 3.17)}$$

  The number of modes is not fixed a priori. Moreover, heuristic procedures are utilised for merging the existing modes (Piccardi, 2004).

  This method is faster than non-parametric kernel density estimation (KDE) and in comparison with KDE, it requires low memory. Also it is faster than the standard mean-shift based estimation.

3.4 A brief comparison of background removal techniques

In this section a brief comparison among the reviewed background removal algorithms in Section 3.3 (subsections 3.3.1 and 3.3.2) is given. The methods reviewed are as follows:

- Non-recursive techniques:
  - Frame differencing
  - Mean filter
  - Mode filter
  - Median filter
  - Basic methods (i.e. mean, mode and median) with selectivity
  - Least median of squares

- Linear predictive filter
- Non-parametric background model or kernel density estimation (KDE)
- Standard mean-shift based estimation
- Eigenbackground Subtraction

➢ Recursive techniques:

- Running average
- Running average with selectivity
- Approximated median filter
- Kalman filter
- Running Gaussian average
- Mixture of Gaussians
- Sequential kernel density approximation (SKDA)

The criteria for comparison among these methods are the 'speed of computations', the 'amount of required memory', and 'accuracy'. Let's compare the above methods based on these criteria and the data available from the literature as follows (Piccardi, 2004):

➢ The speed of computations:

- Real-time:
  - ♦ frame differencing, mean, mode, median (close to real-time), basic methods with selectivity, running average, running average with selectivity, approximated median filter
- Non-real-time:
  - ♦ Almost fast[†]: Kalman filter (depending on the application), running Gaussian average, least median of squares, linear predictive filter
  - ♦ Intermediate: non-parametric background model, mixture of Gaussians
  - ♦ Slow: standard mean-shift based estimation

➢ The amount of required memory:

- High: mean, mode, median, basic methods with selectivity, non-parametric background model, standard mean-shift based estimation, least median of squares
- Intermediate: Kalman filter, running Gaussian average, linear predictive filter, mixture of Gaussians

[†] i.e. frame rates are more than 50% and less than 90% of real-time frame rates.

- Low: frame differencing, running average, running average with selectivity

➢ Accuracy:

Accuracy may be defined based on a few precise measurement metrics such as *TP*, *FP*, *TN*, and *FN* as already defined in Section 2.2.9.4. For computing the accuracy of different background removal methods a number of specified video sequences should be available. Then a researcher should implement each background algorithm and calculate the above measurement metrics for every sequence. This is absolutely a difficult and a time-consuming process since a large number of methods have been proposed in the literature especially in recent years. In addition, it is possible that the authors of some papers do not offer all the details of their algorithms. Thus, the researcher may not implement a method exactly in the same way that its authors have done it. So a little (or sometimes a substantial) different results may be obtained. The original (source or executable) code of an algorithm is often not available as a means of measuring the performance of each method. Thus, a very precise and unbiased comparison among various background algorithms is almost impossible.

Instead of accuracy, the correctness (or the overall performance) of a background algorithm may be defined as how much each method is capable to handle the factors which was given in Section 3.2.3. Table 3.1 is offered for this purpose to present important features of each algorithm.

It is difficult to compare different background removal algorithm since each one has some advantages and disadvantages. However, by looking on Table 3.1 and comparing different methods, someone can decide which background removal algorithm is more appropriate for his application. For example, a researcher may pay little attention to the ‘amount of required memory’ because nowadays, most ordinary PCs have at least 2 to 4GB of RAM. This amount of memory is often sufficient enough for implementing most of background algorithms. Therefore, at the present time, the ‘required memory’ is not considered a limiting constraint (or a bottleneck) for a method. On the other hand, algorithms, which can run in real-time with acceptable performances, may have more importance in many applications. There are very fast PCs at the market, however, a number of sophisticated background algorithms still cannot run in real-time. As another example, an algorithm may be able to handle multimodal backgrounds but it is not capable to manage the problem of the movements of background objects.

Based on the above discussion and depending on the application, researchers can decide which features of a background method are more important to be suitable for their vision system.

Method	Non-Recursive (N) or Recursive (R)	Real-time	Required Memory
Frame Differencing	N	Yes	Low
Mean Filter	N	Yes	High
Mode Filter	N	Yes	High
Median Filter	N	Yes	High
Basic methods with selectivity	N	Yes	High
Least Median of Squares (LMedS)	N	Yes	High
Linear Predictive Filter	N	Yes	Intermediate
Non-parametric Background Model	N	No (Relatively Slow)	High
Standard Mean-Shift Based Estimation	N	No (Very Slow)	High
Eigenbackground Subtraction	N	No (Relatively Slow)	Intermediate
Running Average	R	Yes	Low
Running Average with Selectivity	R	Yes	Low
Approximated Median Filter	R	Yes	High
Kalman Filter	R	No (Almost Fast)	Intermediate
Running Gaussian Average	R	No (Almost Fast)	Intermediate
Mixture of Gaussians (MOG)	R	No (Relatively Slow)	Intermediate
Sequential Kernel Density Approximation	R	No (Relatively Slow)	Intermediate

Table 3.1 – A brief comparison among reviewed background algorithms.

Chapter 4

Selecting Optimum Thresholds for the Colour Difference Image

4.1 Introduction

This chapter is concerned with surveying thresholding methods in order to select optimum thresholds for the colour difference image.

4.2 Optimum thresholding method for colour difference image

Suppose a suitable dynamic background generation algorithm is used to produce an adaptive colour background frame for each colour input frame of an image sequence. Then based on Eq. 2.1, the absolute difference between the current input image $I_i(x, y)$ and its corresponding background frame (i.e. $B_i(x, y)$) for each pixel (x, y) are obtained. In this case, what is the best thresholding method for each colour difference image D_i that satisfies the following conditions?

1. The optimum thresholding method (OTM) should obtain the highest quantitative measurement scores based on PCC, Jaccard and Yule coefficient values where these coefficients are explained in the following.

2. The OTM should produce the best quality binary images in which the shape and the connectivities between the pixels of the foreground regions are preserved and the minimum numbers of spurious regions (which are due to noise) are produced.

3. The OTM should be the fastest thresholding algorithm (among all thresholding methods) which can operate on colour images and is able to convert the colour difference image to binary in a short fraction (e.g. 5% to 10%) of the real-time period. The real-time period is defined as a period of time between every two consecutive frames in an image sequence, which are entered to the

vision system. For example, if 25 frames are input in each second (generated by the frame grabber), based on this definition, the real-time period is equal to 40 ms.

4. The OTM's parameters are automatically computed for each frame.

where PCC (i.e. percentage correct classification), Jaccard (Sneath and Sokal, 1973) and Yule (Sneath and Sokal, 1973) coefficients are defined based on TP (true positives), FP (false positives), TN (true negatives), and FN (false negatives) as follows:

- TP: Number of change pixels correctly detected.
- FP: Number of no-change pixels incorrectly flagged as change by the algorithm.
- TN: Number of no-change pixels correct detected.
- FN: Number of change pixels incorrectly flagged as no-change by the algorithm.

- PCC = (TP + TN) / (TP + FP + TN + FN) (Eq. 4.1)

- Jaccard coefficient = TP / (TP + FP + FN) (Eq. 4.2)

- Yule coefficient = | (TP /(TP + FP)) + (TN/(TN + FN) – 1 | (Eq. 4.3)

Now that the question concerning thresholding is specified, it is time to search for an OTM which satisfies the above conditions. Based on the survey of Sezgin and Sakur (2004), seven thresholding methods with high performances are examined on three video sequences. For this purpose, for every input frame of each three image sequences, the corresponding dynamic background image is computed using the algorithm outlined in ***Chapter 6*** (the 'Selective Update Using Temporal Median'). Then, using Eq. 2.1, the absolute difference D_i is obtained for three colour channels. Table 4.1 shows the PCC, Jaccard and Yule coefficients for seven thresholding methods applied to three video sequences *Fld*, *Bijan1* and *Lab* where these coefficients are computed as follows:

In order to compute the PCC, Jaccard and Yule coefficients for each thresholding method, a number of colour difference images (e.g. 100 frames) of three video sequences *Fld*, *Bijan1* and *Lab* are selected. For simple description of the process, consider for example *Fld* sequence.

For each colour channel *c* of every sample of difference image of *Fld* sequence, the adaptive grey-scale threshold τ_c is computed based on the technique described in the paper of each thresholding method. Then, the grey-scale converter compares the pixels of the difference image with their corresponding grey-scale thresholds τ_c as follows:

$$\text{Grey\_Mask}_c\,(x, y) = \begin{cases} 1, & \text{if difference image}_c\,(x, y) > \tau_c \\ 0, & \text{otherwise} \end{cases} \qquad \text{(Eq. 4.4)}$$

where c = red, green and blue.

After thresholding using one of the methods in Table 4.1, a size filter with specific T_0 value (e.g. $T_0 = 10$) is applied to every sample of thresholded image sequence of *Fld*. Utilising the size filter is necessary because the above coefficients are computed correctly once the video frames are free of spurious small noisy regions.

Algorithm	**Measure**	**Image Sequence**		
		Fld	***Bijan1***	***Lab***
Ridler and Calvard (1978)	***PCC***	**0.9806**	**0.8428**	**0.9558**
	Jaccard	**0.8218**	**0.3638**	**0.7394**
	Yule	**0.9735**	**0.7147**	**0.9269**
Tsai (1995)	***PCC***	**0.9627**	**0.8316**	**0.8683**
	Jaccard	**0.6549**	**0.3160**	**0.4109**
	Yule	**0.9598**	**0.6983**	**0.7715**
Otsu (1979)	***PCC***	**0.9863**	**0.9402**	**0.9600**
	Jaccard	**0.8744**	**0.7815**	**0.8214**
	Yule	**0.9730**	**0.8210**	**0.8864**
Kapur (1985)	***PCC***	**0.9174**	**0.8140**	**0.8843**
	Jaccard	**0.2379**	**0.2354**	**0.2371**
	Yule	**0.9069**	**0.9603**	**0.9240**
Huang and Wang (1995)	***PCC***	**0.8821**	**0.9402**	**0.8867**
	Jaccard	**0.0416**	**0.7815**	**0.1008**
	Yule	**0.8807**	**0.8210**	**0.8759**
Yager (1979)	***PCC***	**0.8966**	**0.7808**	**0.8410**
	Jaccard	**0.0439**	**0.0184**	**0.0317**
	Yule	**0.8961**	**0.7629**	**0.8322**
Unimodal (Rosin, 2001)	***PCC***	**0.9918**	**0.9516**	**0.9649**
	Jaccard	**0.9331**	**0.8300**	**0.8640**
	Yule	**0.9790**	**0.8460**	**0.8899**

Table 4.1 – PCC, Jaccard and Yule coefficients for different threshold methods.

After applying the size filter, a further important point is that how the results of three binary masks of a sample thresholded difference image are combined to produce the final binary frame. For simplicity and speeding up the total thresholding process, the following possibilities are *our proposal* for combining the binary masks. The one, which produces the best result, is selected. Of course, other approaches with better results may exist, which requires more investigation and/or research.

- Initillay, Binary_frame (x, y) $\leftarrow$ 0

1. if (Grey_Mask$_{Red}$ (x, y) = 1 **and** Grey_Mask$_{Green}$ (x, y) = 1 **and**
   Grey_Mask$_{Blue}$ (x, y) = 1) **then** (Eq. 4.5)
   Binary_frame (x, y) $\leftarrow$ 1

2. if (Grey_Mask$_{Red}$ (x, y) + Grey_Mask$_{Green}$ (x, y) + Grey_Mask$_{Blue}$ (x, y) $\geq$ 2) **then**
   // i.e. at least two of three or ***majority*** (Eq. 4.6)
   Binary_frame (x, y) $\leftarrow$ 1

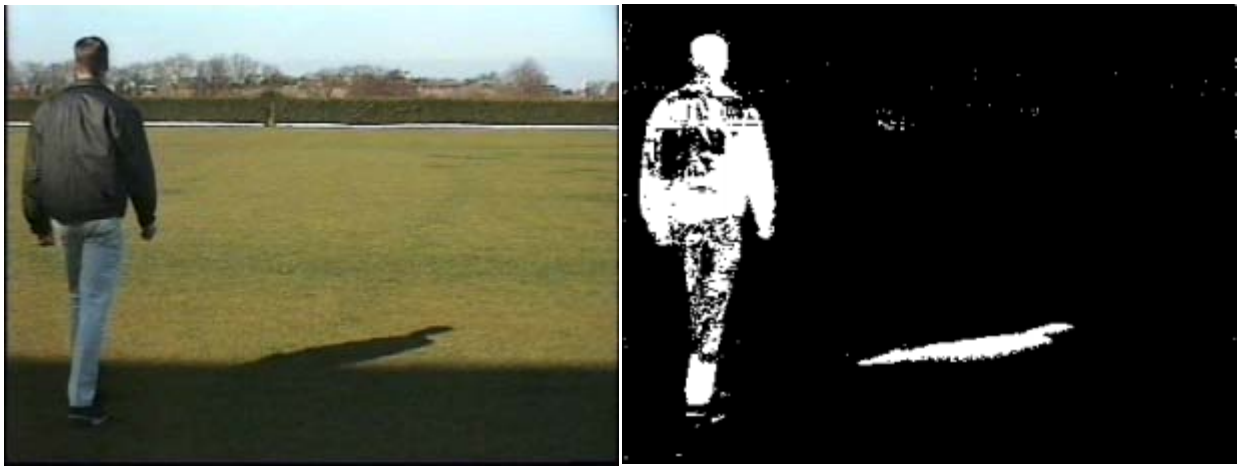
3. if (Grey_Mask$_{Red}$ (x, y) = 1 **or** Grey_Mask$_{Green}$ (x, y) = 1 **or**
   Grey_Mask$_{Blue}$ (x, y) = 1) **then** (Eq. 4.7)
   Binary_frame (x, y) $\leftarrow$ 1

In order to choose the best option, the results of these techniques are qualitatively compared with each other based on a number of images as shown in Figs. 4.1a-l (for three input images Fld249, Highway I-170, Highway II-196). For quantitative comparison, the ground-truth data for true foreground regions should be obtained. Then, the results of each above approach for combining binary masks should be compared with the ground-truth results. However, based on the reasons stated in the following, the results of qualitative comparisons are so clear that the quantitative measurements surely confirm them. Therefore, quantitative results are not necessary.

Figs. 4.1c, 4.1g, and 4.1k are thresholded images produced using the "OR" operation. Due to noise and different threshold values for colour channels, a number of pixels around the foreground regions have also been added to these regions. As a result, the foreground objects have become bigger than their original sizes and their shapes have been distorted. Thus, the "OR" operation is not a suitable selection for combining three binary masks. In contrast, in the thresholded images based on the "AND" operation (Figs. 4.1b, 4.1f, and 4.1j), many pixels inside foreground objects have been removed. So foreground objects have been eroded and their outer boundaries have been distorted as well.

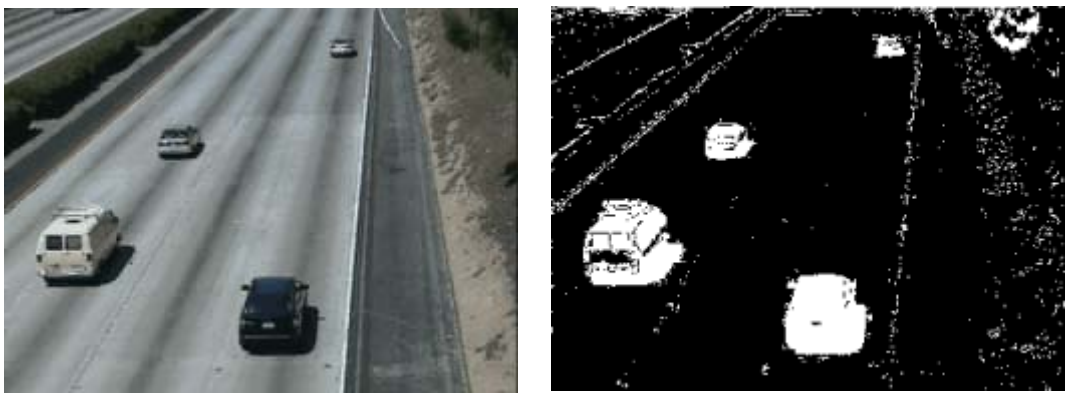
Among three proposed methods for combining three binary masks, the "*majority*" operation (i.e. at least two of three) has suitable performance (Figs. 4.1d, 4.1h, and 4.1l). Because in the thresholded images using the "*majority*" operation, the sizes of foreground objects have not been changed and minor distortions are found in their silhouettes. Thus, it is chosen as the appropriate approach for combining the binary masks to produce the final thresholded difference image. In fact, the PCC, Jackard and Yule coefficients in Table 4.1 for seven thresholding methods have been obtained using the "*majority*" operation. Then, in the resulted binary image, the sizes of all regions are computed. Meanwhile, by looking at that binary frame, a decision

about each region is made whether that region belongs to which one of TP, FP, TN, or FN sets. This approach is repeated for all regions in the binary frame. In practice, regions are shown in different colours based on decreasing sizes for their easier detection in the binary image. Next, the sum of sizes of regions in TP, FP, TN, and FN sets of each frame is obtained. This cumbersome process is performed for all sample frames of Fld video sequence until the sum and average values for the mentioned sets are computed. In this way, the PCC, Jaccard and Yule coefficients are obtained for that thresholding method and the video sequence.



(a) (b)

(c) (d)



(e) (f)

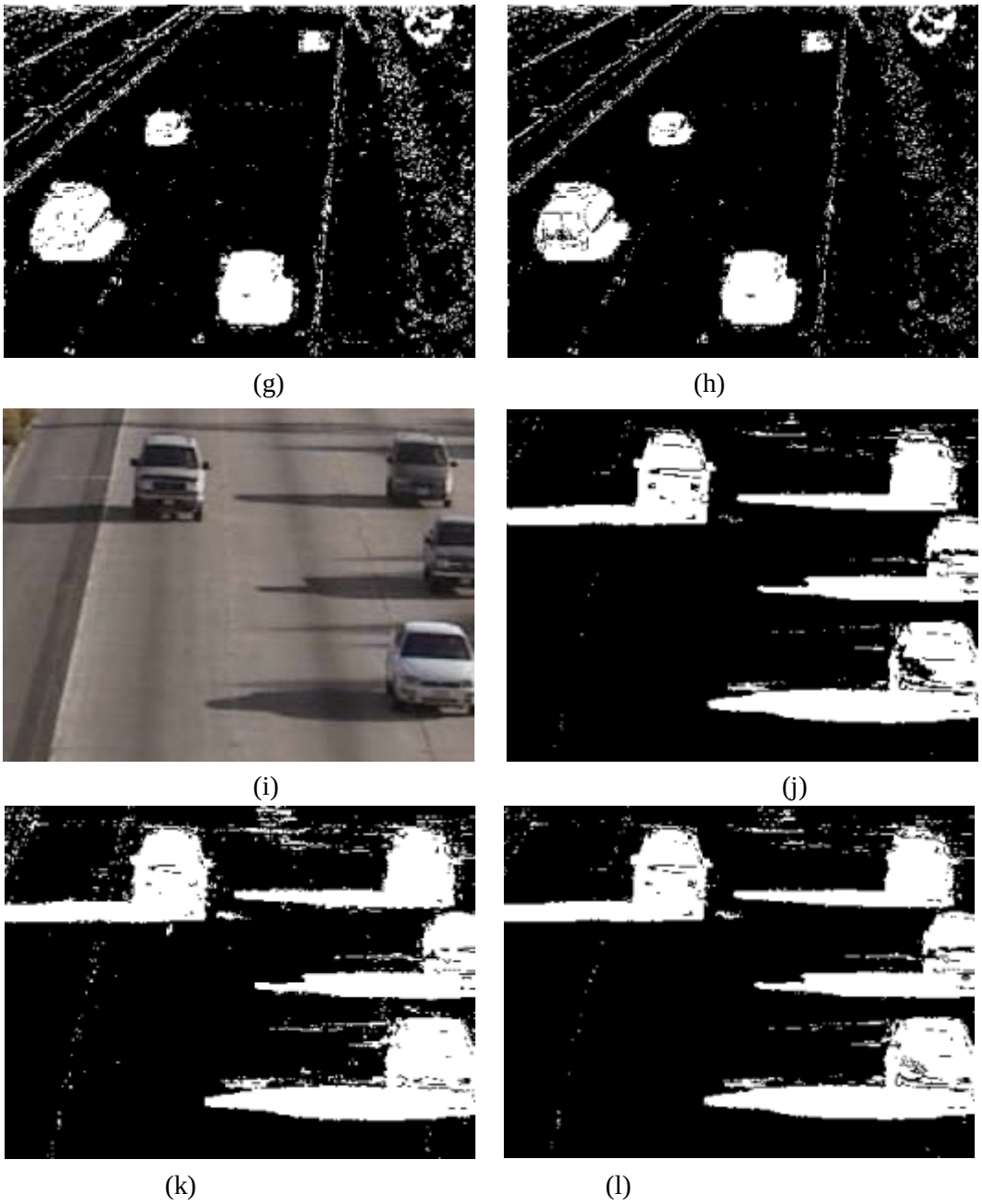


Fig. 4.1 – (a), (e), (i) Fld249, Highway I-170, Highway II-196; (b), (f), (j) using AND thresholding, respectively; (c), (g), (k) using OR, resp.; (d), (h) and (l) using "*majority*", respectively.

Based on the results of Table 4.1, unimodal thresholding has the most appropriate performance. It is a robust technique when the histogram of the grey-scale image has one peak (e.g. in the case of difference image). That is, the intensity of the majority of pixels is very close to the origin. In unimodal thresholding, a straight line is drawn from the peak to the high end of the histogram of the grey-scale image. More precisely, the line starts at the largest bin and finishes at the first empty bin of the histogram following the last filled bin. If the *i*th entry of the histogram is written as H_i then the line $(x_s, y_s) \rightarrow (x_f, y_f)$ is defined as $(\text{argmax}_i\, H_i, \max_i H_i) \rightarrow (\max_{Hi=0\ and}$

$_{Hi-1\neq 0}$ i, 0). The threshold point is selected as the histogram index i that maximises the perpendicular distance between the line and the point (i, H_i); see Fig. 4.2 (Rosin, 2001).

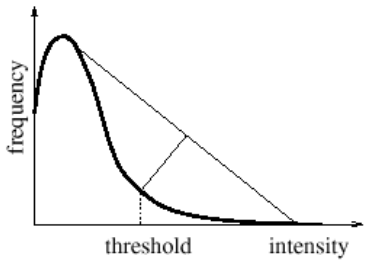


Fig. 4.2 – The procedure for determining threshold from intensity histogram (Rosin, 2001).

As a systematic approach for binarising a colour image, it is first converted to a grey-scale intensity image using, for example, the HSI (H: Hue, S: Saturation, I: Intensity) or YUV (Y: luminance, U: Red-Y, V: Blue-Y) colour models. Then, using any appropriate thresholding method, the grey-scale image is transformed into binary (Du et al., 2004).

Therefore, the following two approaches can be adopted:

1. Using unimodal thresholding, each grey-scale image (i.e. red, green and blue) of the colour difference image is separately converted into binary. Then, these binary images are combined into the final binary frame using the "*majority*" operation as explained above.

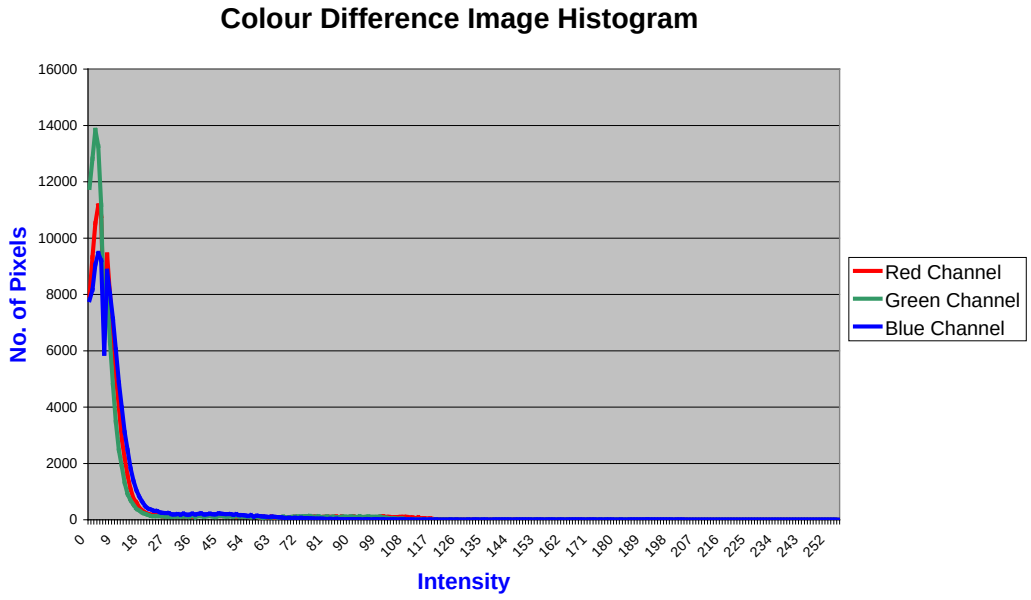
2. The colour difference image is first converted into a grey-scale intensity image. Then, the grey-scale image is transformed into binary using unimodal thresholding (or any other suitable thresholding method).

Which one of the above approaches is more suitable for block number 5 (i.e. grey-scale converter) of Fig. 2.1? Consider the following example as an answer to this question.

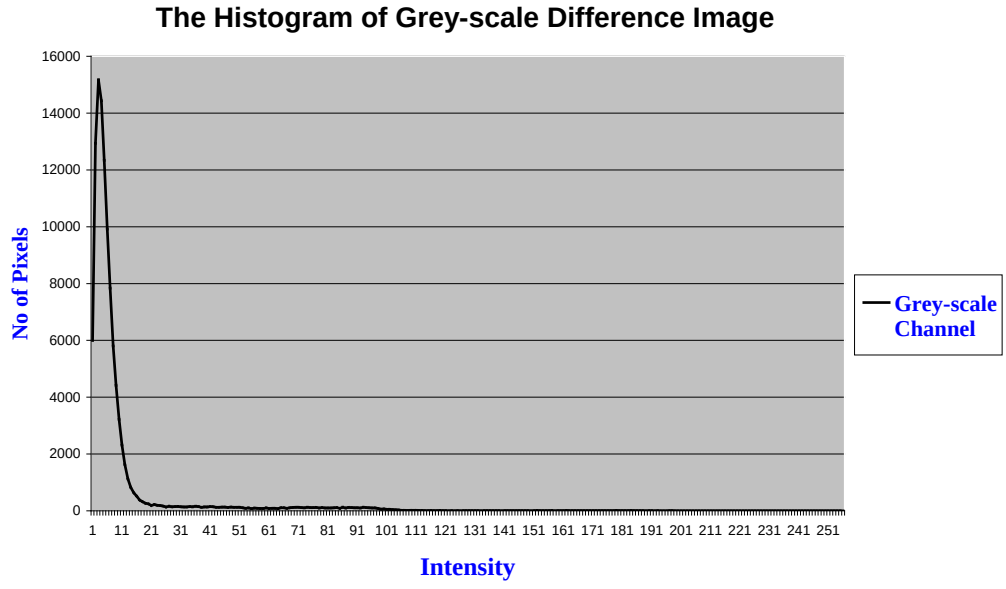
Fig. 4.3 illustrates a sample colour difference image (4.3a), its grey-scale version (4.3b), the histogram of the colour difference image (4.3c), and the histogram of its corresponding grey-scale difference image (4.3d). Fig. 4.3c shows that each colour channel has only one peak. Similarly, the histogram of Fig. 4.3d has also one peak. Thus, unimodal thresholding can be utilised for both colour difference image and its grey-scale version as their results are shown in Fig. 4.4d and Fig. 4.4e, respectively.

For a precise answer to the question mentioned above, both approaches are applied to a number of colour and grey-scale difference images respectively and their binarised outputs are compared with each other qualitatively.

(a) (b)



(c)



(d)

Fig. 4.3 – (a) Fld_Dif249_colour; (b) Fld_Dif249_Grey; (c) Histogram for (a) with Red-τ = 12, Green-τ = 10, Blue-τ = 14; (d) Histogram for (b) with Grey-τ = 15.

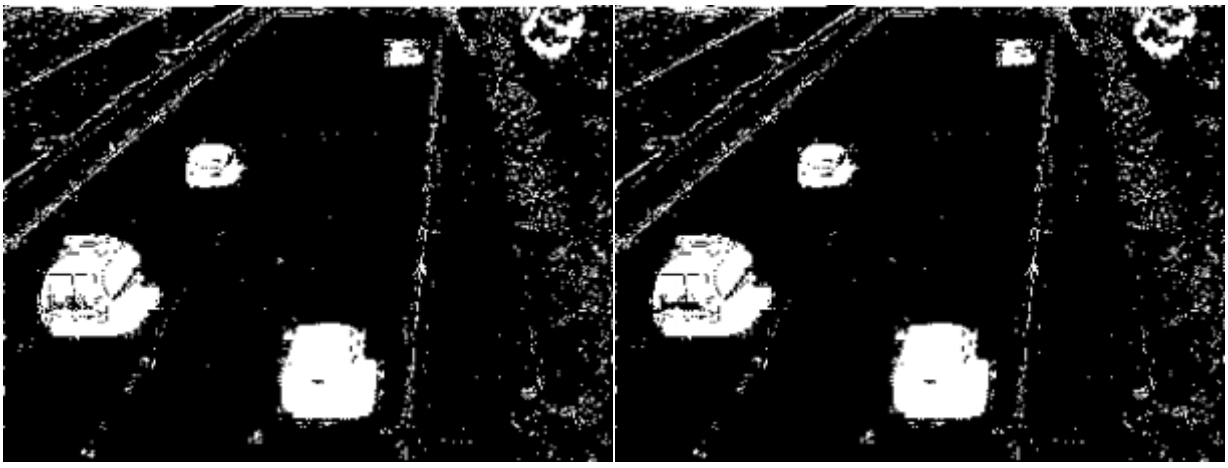
In Fig. 4.4 the binarised colour and grey-scale difference images are shown with their almost actual sizes so that their qualitative comparisons are easily possible. Based on the results of Fig. 4.4, it is observed that binarised colour difference image using the "*majority*" operation are more than 95% similar to binarised grey-scale difference images. The main reason for this is that they seem to be equivalent.

Therefore, either technique can be used. In fact, the grey-scale difference image has been obtained based on the combination of three colour channels.

(a) (b) (c)

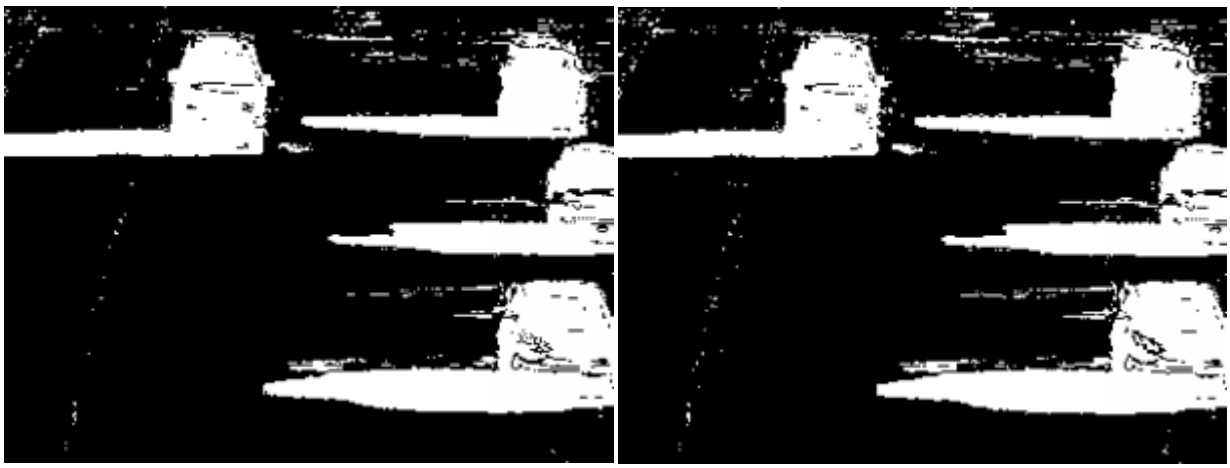
(d) (e)

(f) (g) (h)



(i) (j)

(k) (l) (m)



(n) (o)

Fig. 4.4 – (a) Fld249; (b) Col_Fld_Diff249; (c) Grey_Fld_Diff249; (d) Col_Fld_Thresh249 using "majority" operation; (e) Grey_Fld_Thresh249; (f) HighwayII-170; (g) Col_HighwayII_Diff170; (h) Grey_ HighwayII_Diff170; (i) Col_ HighwayII_Thresh170; (j) Grey_ HighwayII_Thresh170; (k) HighwayI-196; (l) Col_HighwayI_Diff196; (m) Grey_ HighwayI_Diff196; (n) Col_ HighwayI_Thresh196; (o) Grey_ HighwayI_Thresh196.

Chapter 5

A Pixel-based Approach to Adaptive Dynamic Background Subtraction

5.1 Introduction

The importance of producing precise dynamic reference images is obvious in computer vision applications such as video surveillance, traffic monitoring and, human-machine interaction systems, etc. For this purpose, background subtraction has often been used as the first step in many moving object detection algorithms (e.g. Elgammal et al. 1999; Horprasert et al. 1999; Grimson et al. 1998; Kaewtrakulpong and Bowden, 2001; Cucchira et al. 2000 and 2001). By applying a suitable updating technique, an adaptive background subtraction method generates a corresponding reference frame for every input image.

A trivial approach for background estimation is where the reference image is obtained as the scene is static (i.e. there is no background motion). However, since there are variations in lighting conditions caused by changes of light level in an outdoor environment (e.g. due to position changes of the sun, clouds, etc.), the reference frame gets out of date very soon. Thus, adaptive updating techniques should be applied to the background image in order to keep it up-to-date (Rosin and Ellis, 1995).

In a realistic situation, it may sometimes be impractical for a surveillance system to acquire a background image with no moving object (e.g. for a traffic surveillance system which monitors a street or a highway) and sometimes *stationary* objects are moved away from the scene (e.g. a parked car is moved out or a gate is opened and then is left opened) (Dawson-Howe, 1996). An adequate dynamic background generation technique can overcome these problems.

A considerable number of methods concerning adaptive background generation have been proposed since more than twenty years ago. Several papers suggest background update based on statistical temporal functions on a sequence of the most recent frames such as mean (Dagless, et al., 1993), mode (Shio and Slansky, 1991) or

median (Cucchira et al., 2000 and 2001). In a number of other papers, an adaptive parametric or non-parametric mixture model of *k* Gaussian distributions has been used in order to handle small and frequent illumination changes (Elgammal et al., 1999, Stauffer and Grimson, 1999 and Kaewtrakulpong and Bowden, 2001).

The use of Kalman filtering as an adaptive background estimation method has also been proposed by some authors (Stauffer and Grimson, 1999). For each pixel, they use a signal processing system that is controlled by a Kalman filter in order to track the illumination variations of background images. However, some of these methods are unable to manage the problem of moving background objects and foreground objects becoming motionless after a period of time (Koller et al., 1994).

In this chapter, a 'Pixel-based' approach is presented. For this approach, three algorithms called the 'Selective Update Using Temporal Averaging', the 'Selective Update Using Non-Foreground Pixels of the Input Image' and the 'Selective Update Using Temporal Median' are offered and then compared.

In the 'Pixel-based' approach, after thresholding the difference image, background pixels are examined. This check is done to distinguish real background pixels. Then, these pixels are updated using the mentioned 'Selective Update' algorithms. However, there is no processing for foreground pixels except that their corresponding pixels in the previous background image are copied into the current background frame.

5.2 Definitions, characteristics and problems with background generation

The goal of the 'Pixel-based' approach is to produce dynamic background (or reference) images using colour image sequences in unconstrained outdoor and indoor environments. As in Cucchira et al. (2001), the following assumptions are made:

Stationary-background assumption: The camera and the background are stationary.

The following factors change the background:

Variable-lighting assumption: Gradual variation in lighting conditions (some exceptions are, for example, abrupt changes of light level in an outdoor environment due to covering and uncovering the sun by clouds or lights switched on or off in indoor scenes).

Objects-status assumption: The status of objects changes from moving to motionless or from stationary to moving.

This work also supports similar definitions for moving objects and background pixels and the same definition for ghost given in Cucchira et al. (2001) as follows:

Moving-Object-Def: A moving object is a set of connected points in the input image, which in comparison to the static camera, is currently characterised by

non-null motion and a different visual appearance from the background.

Ghost-Def: "Ghost: is a set of connected points, detected as in motion but not corresponding to any real moving object".

Background-Def: Background is all the pixels in every input image which neither belong to moving objects nor ghosts or their cast shadows.

Moving objects, ghosts and their cast shadows are defined as foreground objects (or regions) hereafter. Thus, *Background-Def* is the definition of a part of a dynamic background image which is not occluded by any foreground object. Meanwhile, *Objects-status assumption* and *Ghost-Def* introduce two difficult situations that have important effects on producing dynamic background frames correctly.

5.3 Algorithms for generating dynamic background

There are four groups of pixels that the dynamic background generation system (the background system for short hereafter) should discriminate:

Background-Grp: The pixels that are not occluded by any foreground object during a period of time and often their illuminations gradually change from each input frame to the next. Such pixels usually constitute the majority of the pixels of each input frame.

Foreground-Grp: The pixels that are temporarily occluded by foreground objects in a number of frames.

Stopped-Moving-Grp: The pixels of the stopped objects, which have been moving in previous frames.

Start-Moving-Grp: The pixels of the initial positions of the background objects, which start moving after a period of time.

The background system generates a background image dynamically for each input frame. Based on the assumptions and definitions given in Section 5.2, three algorithms are introduced in the next sections. It will be seen that performances of these algorithms are different only for background pixels (i.e. *Background-Grp*). However, the novelty of these algorithms is due to including a 'Selective Update' method for foreground pixels (i.e. *Foreground-Grp*, *Stopped-Moving-Grp* and *Start-Moving-Grp*).

5.4 The 'Selective Update Using Temporal Averaging'

Temporal averaging, a simple method for producing the reference image, has two main difficulties as follows:

1. If the background image is obtained based on a sequence of initial input images containing no moving objects, it suffers from the problem that it cannot adapt

itself to gradual illumination changes in an outdoor environment. The main reason is that if moving objects appear in the succeeding input images, the simple temporal averaging cannot be used for producing the reference image as is explained below.

2. In addition, temporal averaging produces poor results if the average frame is generated by taking the mean of a long sequence of images consisting moving objects (see Fig. 5.1). However, by exploiting a 'Selective Update' approach it can produce good results converting it to a simple but effective technique.

Let us consider the illumination changes of three colour-channels of two pixels and their corresponding simple temporal averages over several hundred frames. The first pixel (i.e. *Background-Grp-Pixel*) belongs to *Background-Grp* (see Figs. 5.1(b) and 5.2(a)). For such pixels, time averaging is a suitable representation of background image because it follows the light levels of the pixels in *Background-Grp*.

Now consider a pixel from *Foreground-Grp*, i.e. *Foreground-Grp-Pixel*, when is occluded by a foreground object in a number of frames (e.g. from Fld269 to Fld306). In this case, the illumination of *Foreground-Grp-Pixel* has been severely affected (see Figs. 5.1(a) and 5.2(b)).

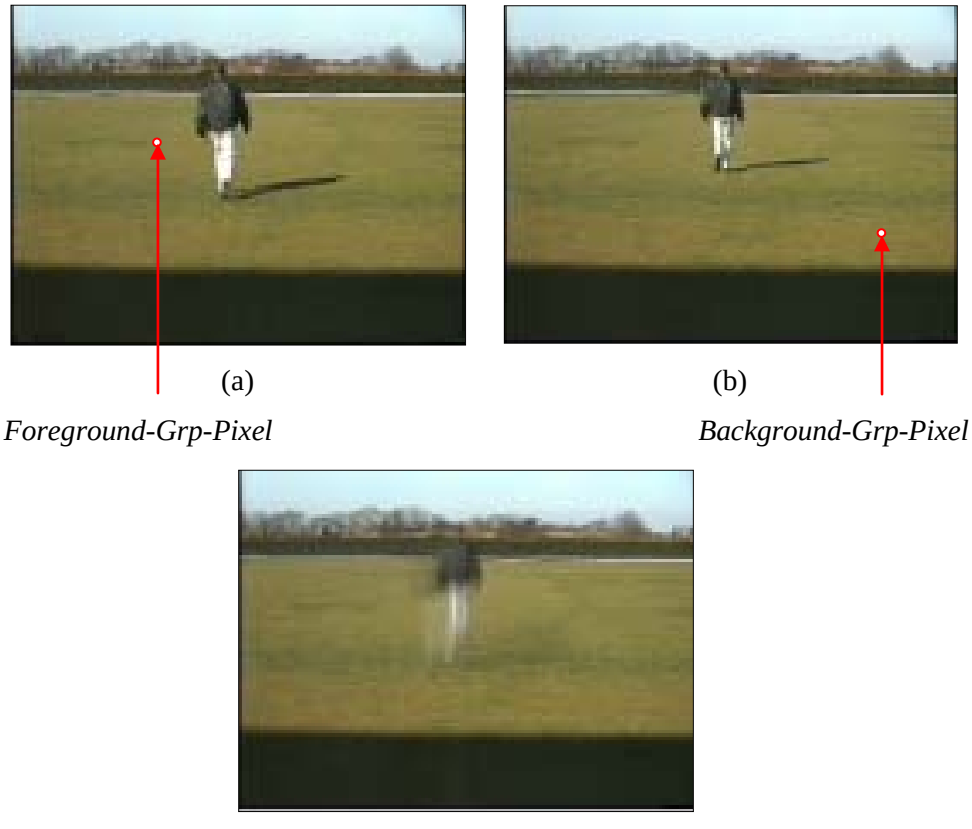
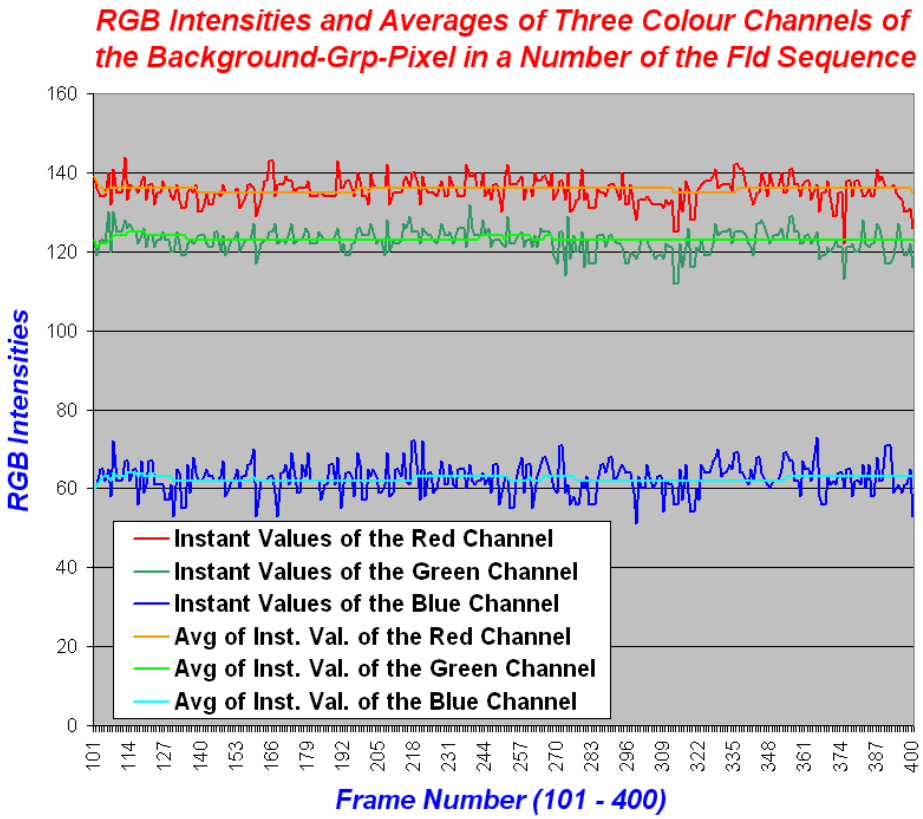
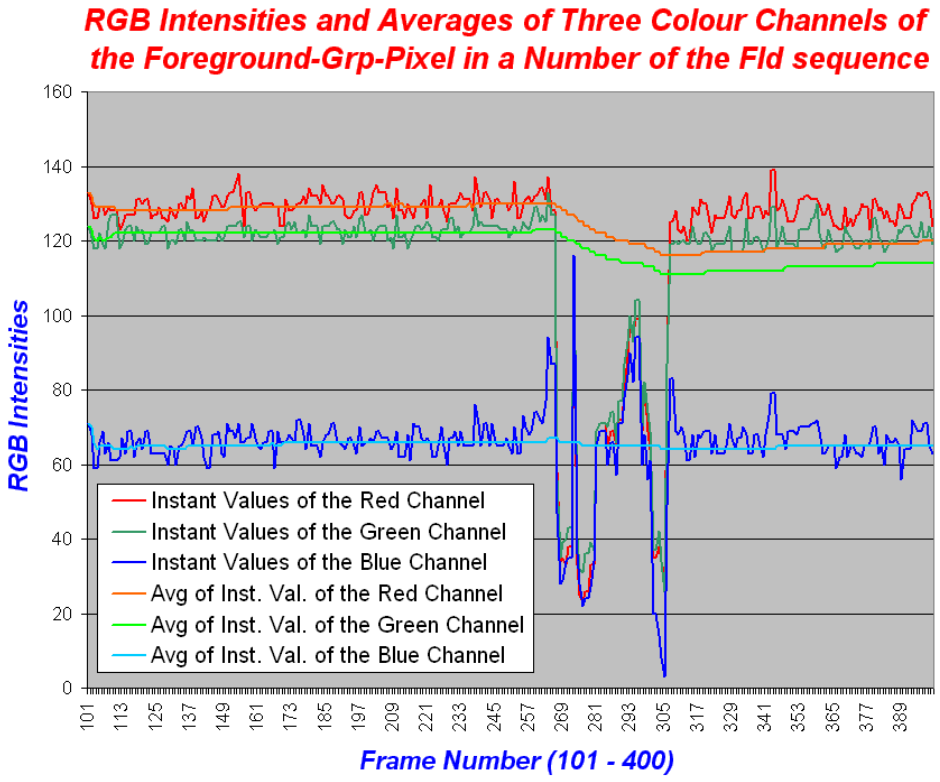


Fig. 5.1 – (a) Input image Fld350 plus the position of *Foreground-Grp-Pixel*, (b) Input image Fld420 plus the position of *Background-Grp-Pixel*, and (c) Temporal Average Temp_Mean420. 'Motion blurring' effect is a result of temporal averaging on a sequence of images (e.g. Fld350 to Fld420) containing moving objects.



(a) RGB intensities and averages of *Background-Grp-Pixel* in the Fld sequence in Fig. 5.1(b)



(b) RGB intensities and averages of *Foreground-Grp-Pixel* in the Fld sequence in Fig. 5.1(a)

Fig. 5.2 – RGB intensities of a pixel from (a) *Background-Grp-Pixel* and (b) from *Foreground-Grp-Pixel* in a number of input images (Fld101 to Fld400) and their corresponding three colour-channel averages.

Based on Fig. 5.2(b), the ‘Selective Update Using Temporal Averaging’ computes temporal averaging for three colour-channels of a pixel (as the background pixel) as long as it is not occluded by a moving object (e.g. for the input frames 101 to 268). Such pixel is a candidate for the background pixels, which are selected for the background update. When the pixel is occluded (e.g. for the input frames 269 to 306), the intensity of the corresponding background pixel remains unchanged by the ‘Selective Update Using Temporal Averaging’. Once again when the pixel in the input image is not occluded by any moving object, the intensity of the background pixel is computed as before the occlusion. So the ‘Selective Update Using Temporal Averaging’ is able to produce correct background frames even though moving objects appear in the input images.

5.5 The ‘Selective Update Using Non-Foreground Pixels of the Input Image’

Based on *Variable-lighting assumption* and *Background-Def*, all the pixels of each input frame, which are currently not occluded by foreground objects, effectively constitute the dynamic background pixels for the ‘Selective Update Using Non-Foreground Pixels of the Input Image’. In fact, the illumination of such pixels in the current input frame differs somewhat from the illumination of the corresponding pixels in the previous input frame. Some examples of exceptions are when in an indoor scene, half of the lamps of a lab are switched on or off. Another example of an exception is when in an outdoor scene the sun is suddenly covered by the clouds or is revealed from behind the clouds. It will be explained in Section 5.7 that it is possible to cope with such special conditions systematically. Thus, for the ‘Selective Update Using Non-Foreground Pixels of the Input Image’, the pixels in *Background-Grp* of each input frame, are selected as the best possible samples for the background pixels. Such pixels are directly included in the current background image.

5.6 The ‘Selective Update Using Temporal Median’

The ‘Selective Update Using Non-Foreground Pixels of the Input Image’ copies the pixels in *Background-Grp* of each input frame in the background image. Instead the ‘Selective Update Using Temporal Median’ copies the median of the last *M* pixels of *Background-Grp* of input images in the reference image. If the number of the pixels in *Background-Grp* of each input frame is large, finding the median can be time-consuming. Thus, suitable techniques such as effective utilisation of histograms must be applied in order to make this method fast enough.

5.7 The ‘Pixel-based’ approach

The pixels in *Foreground-Grp*, *Stopped-Moving-Grp* and *Start-Moving-Grp* plus the exceptions of *Variable-lighting assumption* (i.e. the pixels with relatively large illumination changes) of each input frame, usually greatly differ from the previous

background image. By considering this point, the absolute difference between the current input frame and the previous background image is computed. Then, the appropriate threshold levels for three colour-channels are calculated using unimodal thresholding (Rosin, 2001). If for any pixel in the difference image, two of three colour-channels are higher than their corresponding threshold levels, that pixel is marked as a non-background pixel. The remaining pixels are regarded as candidate background pixels.

The 'Pixel-based' approach considers background pixels after thresholding and updates them using one of the mentioned 'Selective Update' methods. The details of this approach are given below.

After thresholding, the colour features of the candidate background pixels are examined to check whether they really belong to the background image. This check is done to avoid foreground pixels included in the reference frame. For this purpose, the 'Pixel-based' approach applies a colour filter which is implemented using an invariant colour model. In addition, the algorithm is also implemented for the *RGB* space so that a comparison can be made between its results and the invariant colour model. The reason for using an invariant colour model and selecting a suitable colour space for the background algorithm is explained as follows:

Most background subtraction (or removal) algorithms use colour images. Colour image sequences usually have *RGB* format as are produced by frame grabbers. However, as stated in Section 3.2.3, the *RGB* colour space has not a good behaviour with respect to colour perception. Meanwhile, each pixel in the *RGB* colour space has both chromaticity and brightness components. The colours of two pixels are different if either the chromaticity or the brightness of the pixels is different. If the *RGB* colour space is selected for background subtraction, shadows, shadings and highlights are identified as foreground regions even though they only have different brightness but almost the same chromaticity. Thus, it is difficult to remove these lighting effects from foreground regions using only the *RGB* colour space (Hong and Woo, 2003).

Due to above reasons, some background algorithms transform *RGB* colour space to another space such as an invariant colour model (Elgammal et al., 2002, Hong and Woo, 2003, Kampel et al., 2007). In addition, the selection of a suitable colour model is an important issue for the background subtraction algorithm. For this purpose, an invariant colour model should be selected such that it is not only robust against varying illumination in the scene (i.e. due to multiple light sources) but also is robust against changes in the geometry of objects and is also robust against object occlusion and cluttering. Besides, the appropriate colour model should be precise, discriminatory and robust to noise (Gevers and Smeulders, 1999).

A number of invariant colour models have been introduced in the literature including $c_1c_2c_3$, normalised *rgb*, and $l_1l_2l_3$ whose formulae are given in the following (Gevers and Smeulders, 1999):

- $c_1c_2c_3$ model:

$$c_1 = \arctan\left(\frac{R}{\max\{G, B\}}\right) \quad \text{(Eq. 5.1)}$$

$$c_2 = \arctan\left(\frac{R}{\max\{R, B\}}\right) \quad \text{(Eq. 5.2)}$$

$$c_3 = \arctan\left(\frac{B}{\max\{R, G\}}\right) \quad \text{(Eq. 5.3)}$$

- normalised *rgb*:

$$I = R + G + B \quad \text{(Eq. 5.4)}$$

$$r = R / I,\ g = G / I,\ b = B / I, \quad \text{if } I \neq 0 \quad \text{(Eq. 5.5)}$$

$$r = g = b = 0, \quad \text{if } I = 0 \quad \text{(Eq. 5.6)}$$

Only two of these normalised channels are required since $r + g + b = 1$ ($I \neq 0$).

- $l_1 l_2 l_3$ model:

$$l_1 = \frac{(R - G)^2}{(R - G)^2 + (R - B)^2 + (G - B)^2} \quad \text{(Eq. 5.7)}$$

$$l_2 = \frac{(R - B)^2}{(R - G)^2 + (R - B)^2 + (G - B)^2} \quad \text{(Eq. 5.8)}$$

$$l_3 = \frac{(G - B)^2}{(R - G)^2 + (R - B)^2 + (G - B)^2} \quad \text{(Eq. 5.9)}$$

By assuming dichromatic reflection and white illumination, it is shown that $c_1c_2c_3$, normalised *rgb*, and $l_1l_2l_3$ are all invariant with respect to changes in viewing direction, object geometry and scene illumination (refer to Gevers and Smeulders, 1999).

The algorithm of the 'Pixel-based' approach is presented using $c_1c_2c_3$ colour model. Obviously for utilising other colour models mentioned above, the same algorithm is implemented using the components of that model.

The colour filter compares the colour components of the candidate background pixels in the current input frame with the colour components of the corresponding pixels in two previous background frames to decide whether those pixels belong to *Background-Grp* or not. First of all, the algorithm of the 'Pixel-based' approach should be implemented such that there is no limitation for the first input image containing moving foreground objects. Thus, at the beginning, the first input image is considered to be the first background image as well (i.e. $B_{t-1} \leftarrow I_t$). After that, the invariant colour components (i.e. c_1, c_2 and c_3) of candidate background pixels in the current input frame (i.e. I_{t+1}) are computed using Eqs. 5.1 to 5.3. Then, for generating the second previous background frame, they are compared with the invariant colour components of corresponding pixels in a number of preceding input frames. Examples of this are the ninth and the tenth previous input frames. It is assumed that in comparison with a number of preceding input frames, objects have made enough movement and there are no overlaps between their current positions and their previous positions. If this assumption is not right, those two images should be

considered from a large number of preceding frames. Once the previous and second previous background images were obtained, the relative variations (Eq. 5.10) of these pixels are computed as follows:

$$|c_i - c_i^*| / c_i^* \text{ and } |c_i - c_i^{**}| / c_i^{**}, \quad \text{for i = 1, 2, 3} \qquad \text{(Eq. 5.10)}$$

where c_i, c_i^*, c_i^{**} are the colour components of the current input frame, the previous and second previous background frames, respectively. If at least half of them are less than or equal to Low-Threshold (e.g. 3%) and the remaining ones are less than a Mid-Threshold (e.g. 5%), then those pixels are considered as background pixels (i.e. *Background-Grp*). This is called as the *similarity measure* of the relative variations of c_i with respect to c_i^* and c_i^{**}. Then, every input image is compared with previous and second previous background images based on the similarity measure for producing the third and succeeding reference frames.

The next step is classifying non-reference pixels in *Foreground-Grp, Stopped-Moving-Grp* or *Start-Moving-Grp*. In order to check that a pixel is in motion or remains in a position for a period of time, a good technique is to count the number of frames that the pixel remains in that state. Thus, a timer is associated with each pixel in the image. If a pixel of an input image is regarded as a non-reference pixel, its associated timer is incremented. As in Cucchira et al. (2001), a trade-off between high responsiveness to changes of input frames and reliability of the background images computed should be made. For considering a high responsiveness of the background system, two seconds (equal to 50 frames for a vision system with 25 frames/s) is suggested to be a suitable response time. The selection of the background system's response time is an important matter which more information concerning its selection is given later in this section.

The pseudo code of the 'Pixel-based' approach in Fig. 5.3 demonstrates how dynamic reference images are computed based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' as follows:

```
// LT = the Lowest upper Threshold; its value is determined based on the suggested
// background system's response time

LT ← 50;

// The first input image is regarded as the first reference image.

B_st_index (x, y) ← I_st_index (x, y);   // st_index = the start frame number of the sequence
                                         // en_index = the end frame number of the sequence
for (t ← st_index + 1; t ≤ en_index; t ← t + 1)
{
        D_t (x, y) ← |I_t (x, y) - B_t-1 (x, y)|;   // D_t = difference image for input frame I_t

        compute unimodal thresholding (D_t);

        if (two of three colour-channels of a pixel in D_t ≥ corresponding
            threshold levels)
```

```
{
    /* The timer of each non-reference pixel is incremented. However, often
    the pixels of a moving object (i.e. the pixels in Foreground-Grp) may
    occlude the reference pixels for a number of frames. The timers of such
    pixels increase but they usually do not become larger than LT. Thus, the
    background image retains the last input pixel values and it does not
    change. */

    pixel_timer (x, y) ← pixel_timer (x, y) + 1;
    pixel_flag (x, y) ← 0;   // 0: a foreground pixel

    /* If the pixels of a moving object remain in a position for more than
    LT input frames (i.e. Stopped-Moving-Grp, Start-Moving-Grp or excep-
    tions of Variable-lighting assumption), those pixels are included in the
    background image after LT frames. */

    if (pixel_timer (x, y) ≥ LT)
    {
        pixel_timer (x, y) ← 0;
        pixel_flag (x, y) ← 2;   // 2: a possible ghost pixel
    }
}
else
{
    /* Most of the pixels of every input frame that somewhat differ from
    the pixels in the corresponding positions in the previous frame (i.e.
    Background-Grp), constitute the majority of the pixels in the dynamic
    background image. However, it should be examined that their colour
    components are similar to colour components of the corresponding pixels
    in the previous and second previous background frames. */

    compute c1, c2, c3 for the candidate background pixels of the current
    input frame;

    compute c1, c2, c3 for the corresponding pixels in the first previous
    background frame (i.e. prev1_c1, prev1_c2 and prev1_c3);

    compute c1, c2, c3 for the corresponding pixels in the second previous
    background frame (i.e. prev2_c1, prev2_c2 and prev2_c3);

    if (ci has the similarity measure with respect to to prev1_ci and
        prev2_ci)          // i = 1, 2, 3
    {
        pixel_timer (x, y) ← 0;
        pixel_flag (x, y) ← 1;   // 1: a background pixel
    }
    else
    {
        pixel_timer (x, y) ← pixel_timer (x, y) + 1;
        pixel_flag (x, y) ← 0;   // 0: a foreground pixel
```

```
        }
    }

    if (pixel_flag (x, y) = 1)   // i.e. a background pixel
        Bt (x, y) ← It (x, y);    // for the 'Selective Update Using Non-Foreground
                                  // Pixels of the Input Image'

        /* the code for the 'Selective Update Using Temporal Averaging' or the
        'Selective Update Using Temporal Median' goes here in place of the
        'Selective Update Using Non-Foreground Pixels of the Input Image' for
        updating the pixels in Background-Grp. */

    else if (pixel_flag (x, y) = 2)     // a ghost pixel
    {
            Bt (x, y) ← It (x, y);
            pixel_timer (x, y) ← 0;
            pixel_flag (x, y) ← 1;   // i.e. a background pixel
    }
    else
            Bt (x, y) ← Bt-1 (x, y);   // a foreground pixel
} // end of for
```

Fig. 5.3 – The pseudo code of the 'Pixel-based' approach based on the 'Selective Update Using Non-Foreground Pixels of the Input Image'.

The 'Pixel-based' approach can be evaluated from quantitative and qualitative points of view as stated in the next section.

5.8 Evaluation of the 'Pixel-based' approach

In order to determine the effectiveness of the 'Pixel-based' approach, this algorithm is implemented using four colour models and based on three 'Selective Update' methods. Their results are compared in the following subsections.

5.8.1 Quantitative evaluation of the 'Pixel-based' approach

For quantitative evaluation, the algorithm in Fig. 5.3 is also implemented based on the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median'. In addition, the colour filter of each 'Selective Update' method is implem-ented using $c_1c_2c_3$, $l_1l_2l_3$, normalised *rgb* (*Nrgb*) and the *RGB* colour space. The results of applying three 'Selective Update' methods and four colour models to *Fld*, *Bijan1*, *Highway I* and *Lab* video sequences are offered in Tables 5.1 to 5.3. Three 'Selective Update' methods of the 'Pixel-based' approach are implemented on a 2.4 GHz Pentium 4 PC with 1 GB of RAM running windows XP professional edition. Quantitative results are obtained based on the same threshold values for

Low-Threshold, Mid-Threshold, relative variations and the same sequences of frames. In Tables 5.1 to 5.3, the percentage of recognised background pixels (column 8 in Table 5.1, for example) is calculated by dividing total number of recognised background pixels (column 7 in Table 5.1) to total number of reference pixels after thresholding (column 5 in Table 5.1).

Quantitative comparisons are classified based on the colour model and the 'Selective Update' method using the data given in Tables 5.1 to 5.3 as follows:

- Based on the colour model:
  - In all 'Selective Update' methods, $l_1l_2l_3$ has the worst performance among the four colour models irrespective of the utilised video sequence, i.e., *Fld*, *Bijan1*, *Highway I* and *Lab*. In addition, the computations of $l_1l_2l_3$ components are almost time-consuming due to relatively complex components (Eqs. 5.7 to 5.9). Meanwhile, among the four colour models, $l_1l_2l_3$ has the second slowest average processing time (column 4 in Tables 5.1 to 5.3). As a result, $l_1l_2l_3$ model has neither a good performance nor a fast processing time regardless of the applied video sequence or the utilised 'Selective Update' method. Thus, $l_1l_2l_3$ is not quantitatively a suitable colour model for the 'Pixel-based' approach.
  - *RGB* has almost the fastest average processing time among four colour

Sequence Name (W x H)	Applied Colour Model	Utilised No. of Frames	Average Processing Time (ms)	Total No. of Reference Pixels after Thresholding	Total No. of Recognised Non-Background Pixels	Total No. of Recognised Background Pixels	The Percentage of Recognised Background Pixels
Fld (384 x 288)	$c_1c_2c_3$	245	210	25038195	3876476	21161719	84.5
	$l_1l_2l_3$		129	25542849	18056729	7486120	29.3
	Nrgb		113	25176427	4177998	20998429	83.4
	RGB		99	24265496	6169997	18095499	74.6
Bijan1 (352 x 288)	$c_1c_2c_3$	230	215	21802015	1939989	19862026	91.1
	$l_1l_2l_3$		130	21516706	19273725	2242981	10.4
	Nrgb		108	21802214	1492019	20310195	93.2
	RGB		106	21649056	3753364	17895692	82.7
Highway I (320 x 240)	$c_1c_2c_3$	255	117	14657239	228574	14428665	98.4
	$l_1l_2l_3$		62	14816430	9521514	5294916	35.7
	Nrgb		50	14633751	191254	14442497	98.7
	RGB		52	15314984	2913323	12401661	80.1
Lab (320 x 240)	$c_1c_2c_3$	265	138	17965738	195388	17770350	98.9
	$l_1l_2l_3$		89	18539663	6448229	12091434	65.2
	Nrgb		61	17996350	257506	17738844	98.6
	RGB		55	17974508	607813	17366695	96.6

Table 5.1 – The results of applying the 'Pixel-based' approach based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' utilising four colour models to *Fld*, *Bijan1*, *Highway I*, and *Lab* video sequences.

Sequence Name	Applied Colour Model	Utilised No. of Frames	Average Processing Time (ms)	Total No. of Reference Pixels after Thresholding	Total No. of Recognised Non-Background Pixels	Total No. of Recognised Background Pixels	The Percentage of Recognised Background Pixels
Fld	$c_1c_2c_3$	245	223	25740115	3687644	22052471	85.7
	$l_1l_2l_3$		118	25668066	17014466	8653600	33.7
	Nrgb		101	25727703	3709231	22018472	85.6
	RGB		98	25727570	7812724	17914846	69.6
Bijan1	$c_1c_2c_3$	230	190	19939697	2475815	17463882	87.6
	$l_1l_2l_3$		101	20206159	17940489	2265670	11.2
	Nrgb		87	19941384	1813412	18127972	90.9
	RGB		86	19997173	6014812	13982361	69.9
Highway I	$c_1c_2c_3$	255	132	16133388	335352	15798036	97.9
	$l_1l_2l_3$		66	14504233	11435165	3069068	21.2
	Nrgb		57	16130117	260886	15869231	98.4
	RGB		60	15272052	9336630	5935422	38.9
Lab	$c_1c_2c_3$	265	153	18941310	236349	18704961	98.8
	$l_1l_2l_3$		98	18565924	9063777	9502147	51.2
	Nrgb		71	19015478	286310	18729168	98.5
	RGB		58	19009145	1039614	17969531	94.5

Table 5.2 – The results of applying the 'Pixel-based' approach based on the 'Selective Update Using Temporal Averaging' utilising four colour models to *Fld*, *Bijan1*, *Highway I*, and *Lab* video sequences.

Sequence Name	Applied Colour Model	Utilised No. of Frames	Average Processing Time (ms)	Total No. of Reference Pixels after Thresholding	Total No. of Recognised Non-Background Pixels	Total No. of Recognised Background Pixels	The Percentage of Recognised Background Pixels
Fld	$c_1c_2c_3$	245	276	25742063	3944640	21797423	84.7
	$l_1l_2l_3$		156	25651678	17569076	8082602	31.5
	Nrgb		152	25727011	4425648	21301363	82.8
	RGB		142	25722413	7527995	18194418	70.7
Bijan1	$c_1c_2c_3$	230	225	19262166	3019926	16242240	84.3
	$l_1l_2l_3$		136	19735753	17762336	1973417	10.0
	Nrgb		129	19271146	2217875	17053271	88.5
	RGB		120	19136581	4830171	14306410	74.8
Highway I	$c_1c_2c_3$	255	163	15731239	188267	15542972	98.8
	$l_1l_2l_3$		94	11879520	10791442	1088078	9.2
	Nrgb		92	15732572	132721	15599851	99.2
	RGB		87	15694623	2453348	13241275	84.4
Lab	$c_1c_2c_3$	265	193	19024063	358700	18665363	98.1
	$l_1l_2l_3$		114	18288508	12164993	6123515	33.5
	Nrgb		111	19023479	459329	18564150	97.6
	RGB		96	19001963	841610	18160353	95.6

Table 5.3 – The results of applying the 'Pixel-based' approach based on the 'Selective Update Using Temporal Median' utilising four colour models to *Fld*, *Bijan1*, *Highway I*, and *Lab* video sequences.

spaces due to no conversion of its components. However, the invariant colour models $c_1c_2c_3$ and normalised *rgb* often have much higher performances by recognising more numbers of reference and background pixels after thresholding and higher percentages of recognised background pixels (columns 5, 7 and 8 in Tables 5.1 to 5.3). This result is valid for the utilised video sequences and three 'Selective Update' methods. Thus, the data given in Tables 5.1 to 5.3 confirm superiority of the invariant colour models $c_1c_2c_3$ and normalised *rgb* for implementing the 'Pixel-based' approach to the *RGB* colour space.

- Finally, for all applied video sequences and three 'Selective Update' methods, normalised *rgb* has almost the same or a little bit better performance to $c_1c_2c_3$ colour model. However, in comparison with $c_1c_2c_3$ model, normalised *rgb* always has much faster processing time. This is because trigonometric functions are used for the formulas of the components of $c_1c_2c_3$ model (Eqs. 5.1 to 5.3) while the computations of normalised *rgb* components require simple divisions (Eqs. 5.4 to 5.6).

As a result, normalised *rgb*, which has very high performance and fast processing time in Tables 5.1 to 5.3, is the most appropriate colour model for implementing the 'Pixel-based' approach. Therefore, in the following, quantitative comparison based on the utilised 'Selective Update' method is only performed using normalised *rgb*. For the simplicity of comparison, the results of applying three 'Selective Update' methods in Tables 5.1 to 5.3 concerning normalised *rgb*, are summarised in Table 5.4.

- Based on the 'Selective Update' method:

  - Except for *Bijan1* video sequence, the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' recognised more numbers of reference and background pixels after thresholding than 'Selective Update using Non-Foreground Pixels of the Input Image' (columns 4 and 6 in Table 5.4).

  - The percentages of recognised background pixels by the 'Selective Update Using Non-Foreground Pixels of the Input Image' are almost the same or slightly higher than the percentages of the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' for the utilised video sequences (column 7 in Table 5.4).

By considering the above results, the 'Selective Update Using Non-Foreground Pixels of the Input Image' produced slightly higher percentages of recognised background pixels (column 7 in Table 5.4). On the other hand, the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' often recognised more numbers of reference and background pixels after thresholding. It shows that they produce more precise background images because they probably identify fewer numbers of foreground pixels incorrectly as background pixels. As a result, background frames with better qualities are produced by the latter methods, which are due to their filtering effects. In total, depending on the applied video sequence, the 'Selective Update Using Non-Foreground Pixels of the Input

Image', the 'Selective Update Using Temporal Averaging' or the 'Selective Update Using Temporal Median' utilising norm-alised *rgb* may have a slightly better performance. Thus, from quantitative point of view, a deterministic decision cannot be made regarding which 'Selective Update' method has a better performance. However, it is shown in Section 5.9 that the 'Selective Update Using Non-Foreground Pixels of the Input Image' has some deficiencies which make it an inappropriate method for 'Pixel-based' approach.

5.8.2 Qualitative evaluation of the 'Pixel-based' approach

The results of applying each 'Selective Update' method and four colour models to *Fld*, *Bijan1*, *Highway I*, and *Lab* video sequences are shown in Figs. 5.4 to 5.15. For each 'Selective Update' method, each video sequence (e.g. *Fld*) and colour model, thresholded, foreground and background frames corresponding to each input image are shown, respectively. The qualities of all corresponding thresholded, foreground and background frames of each video sequence are almost the same. Thus, only samples of them are offered in Figs. 5.4 to 5.15. Meanwhile, in each foreground frame, pixels recognised by the utilised algorithm as foreground pixels (shown in

Sequence Name	Selective Update Method	Average Processing Time (ms)	Total No. of Reference Pixels after Thresholding	Total No. of Recognised Non-Background Pixels	Total No. of Recognised Background Pixels	The Percentage of Recognised Background Pixels
Fld	SUNFPII	113	25176427	4177998	20998429	83.4
	SUTA	101	25727703	3709231	22018472	85.6
	SUTM	152	25727011	4425648	21301363	82.8
Bijan1	SUNFPII	108	21802214	1492019	20310195	93.2
	SUTA	87	19941384	1813412	18127972	90.9
	SUTM	129	19271146	2217875	17053271	88.5
Highway I	SUNFPII	50	14633751	191254	14442497	98.7
	SUTA	57	16130117	260886	15869231	98.4
	SUTM	92	15732572	132721	15599851	99.2
Lab	SUNFPII	61	17996350	257506	17738844	98.6
	SUTA	71	19015478	286310	18729168	98.5
	SUTM	111	19023479	459329	18564150	97.6

Table 5.4 – The summary of the results of applying the 'Pixel-based' approach based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' (SUNFPII), the 'Selective Update Using Temporal Averaging' (SUTA) and the 'Selective Update Using Temporal Median' (SUTM) to four video sequences concerning normalised *rgb*, which were already given in Tables 5.1 to 5.3.

white pixels) are superimposed on the corresponding input image. In this way, a foreground frame is an indication of how much the algorithm has been successful in recognising foreground and background pixels correctly. In other words, a foreground frame also shows the preciseness and the quality of the corresponding background image produced by the utilised algorithm. The results are as follows:

- Sample frames of the 'Pixel-based' approach based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' (Figs. 5.4 to 5.7):

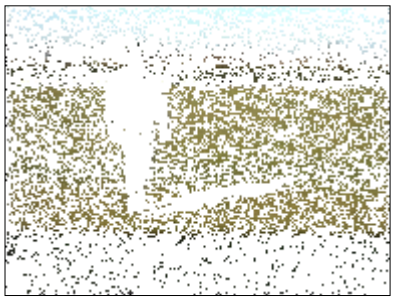
(a) Input image Fld290

(b) $c_1c_2c_3$_Thresh_Fld290

(c) $c_1c_2c_3$_Forgnd_Fld290

(d) $c_1c_2c_3$_Backgnd_Fld290

(e) $l_1l_2l_3$_Thresh_Fld290



(f) $l_1l_2l_3$_Forgnd_Fld290

(g) $l_1l_2l_3$_Backgnd_Fld290

(h) *Nrgb*_Thresh_Fld290

(i) *Nrgb*_Forgnd_Fld290

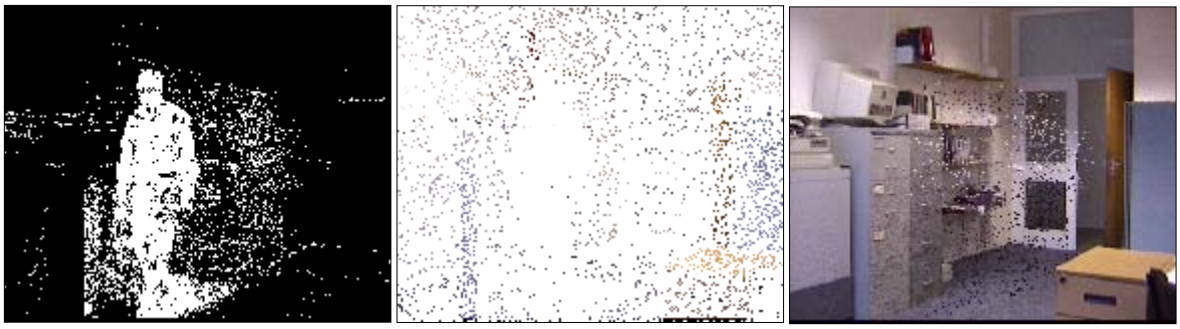
(j) *Nrgb*_Backgnd_Fld290

(k) *RGB*_Thresh_Fld290 (l) *RGB*_Forgnd_Fld290 (m) *RGB*_Backgnd_Fld290

Fig. 5.4 – (a) The input image Fld290; The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' to Fld290 using $c_1c_2c_3$ ((b)-(d)), $l_1l_2l_3$ ((e)-(g)), normalised *rgb* (*Nrgb*) ((h)-(j)), colour models and the *RGB* colour space ((k)-(m)).

(a) Input image Bijan1_262

(b) $c_1c_2c_3$_Thresh_Bijan1_262 (c) $c_1c_2c_3$_Forgnd_Bijan1_262 (d) $c_1c_2c_3$_Backg_Bijan1_262



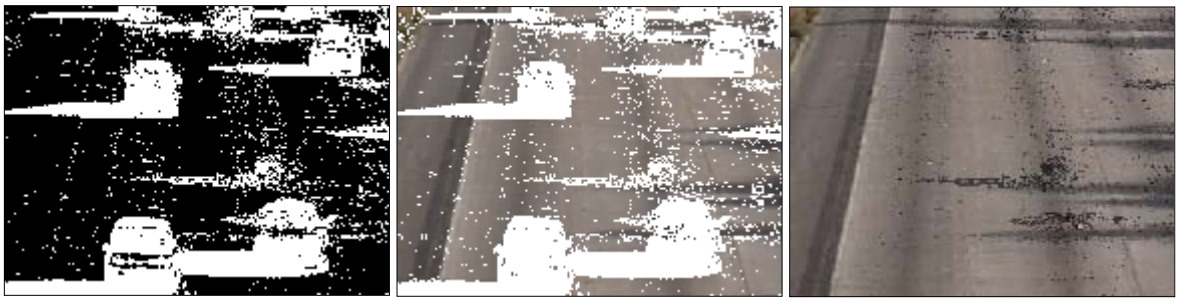
(e) $l_1l_2l_3$_Thresh_Bijan1_262 (f) $l_1l_2l_3$_Forgrnd_Bijan1_262 (g) $l_1l_2l_3$_Backg_Bijan1_262

(h) *Nrgb*_Thresh_Bijan1_262 (i) *Nrgb*_Forgrnd_Bijan1_262 (j) *Nrgb*_Backg_Bijan1_262

(k) *RGB*_Thresh_Bijan1_262 (l) *RGB*_Forgrnd_Bijan1_262 (m) *RGB*_Backg_Bijan1_262

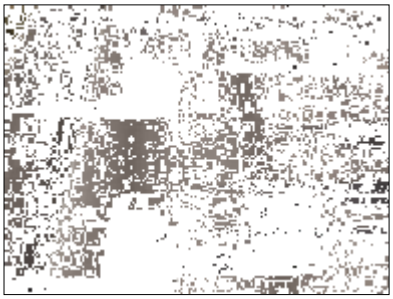
Fig. 5.5 – (a) The input image Bijan1_262; The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' to Bijan1_262 using $c_1c_2c_3$ ((b)-(d)), $l_1l_2l_3$ ((e)-(g)), normalised *rgb* (*Nrgb*) ((h)-(j)), colour models and the *RGB* colour space ((k)-(m)).

(a) Input image Highway I_261



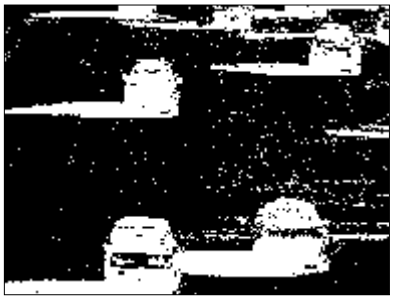
(b) $c_1c_2c_3$_Thrsh_Highway I_261 (c) $c_1c_2c_3$_Forg_Highway I_261 (d) $c_1c_2c_3$_Back_Highway I_261

(e) $l_1l_2l_3$_Thrsh_Highway I_261

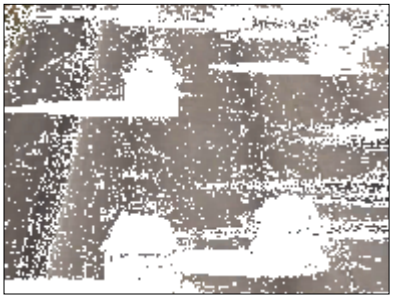


(f) $l_1l_2l_3$_Forg_Highway I_261

(g) $l_1l_2l_3$_Back_Highway I_261

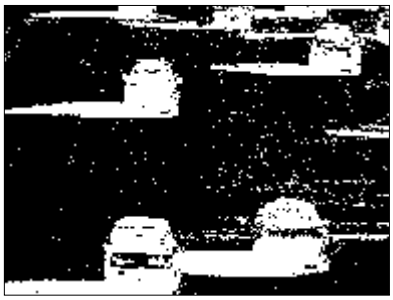


(h) *Nrgb*_Thrsh_Highway I_261

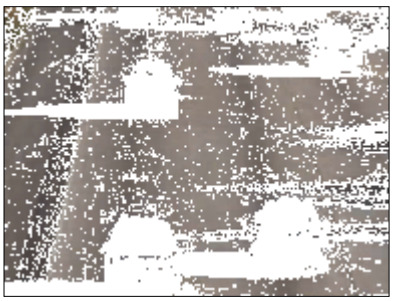


(i) *Nrgb*_Forg_Highway I_261

(j) *Nrgb*_Back_Highway I_261



(k) *RGB*_Thrsh_Highway I_261



(l) *RGB*_Forg_Highway I_261

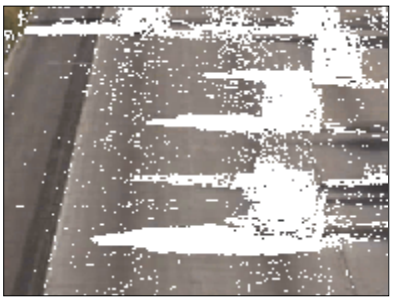
(m) *RGB*_Back_Highway I_261

(n) Input image Highway I_71

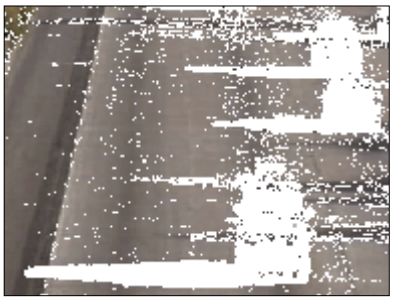
(o) $c_1c_2c_3$_Forg_Highway I_71

(p) $c_1c_2c_3$_Forg_Highway I_85

(q) $c_1c_2c_3$_Forg_Highway I_139



(r) $c_1c_2c_3$_Forg_Highway I_191

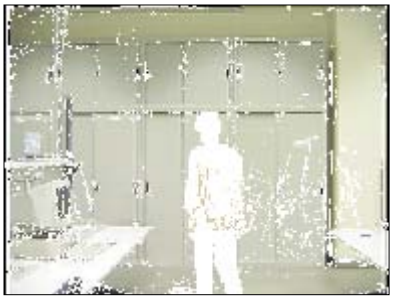


(s) $c_1c_2c_3$_Forg_Highway I_232

Fig. 5.6 – (a) The input image Highway I_261; The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' to Highway I_261 using $c_1c_2c_3$ ((b)-(d)), $l_1l_2l_3$ ((e)-(g)), normalised *rgb* (*Nrgb*) ((h)-(j)), colour models and the *RGB* colour space ((k)-(m)). (n) The initial input image Highway I_71 is used as the first image of the sequence and also as the first background image. (o) Its corresponding foreground image. The 'Selective Update Using Non-Foreground Pixels of the Input Image' considers the moving objects in the first background image as ghosts. However, occasionally there are some overlaps between ghosts and other objects or their cast shadows in less than LT frames in succeeding images. In this case, it may take a large number of frames for the algorithm to remove the pixels of a ghoast (Figs. (p) to (s)). But the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' utilise long buffers (more than LT frames) for their computations. Due to their filtering effect, there is a little possibility that the pixels of ghosts appear in the first background image (see Figs. 5.10 and 5.14).

(a) Input image Lab220

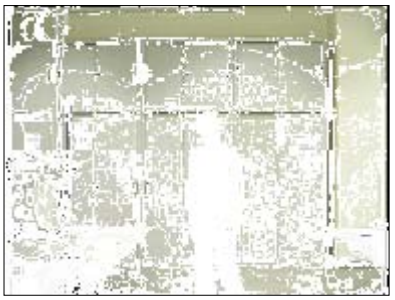
(b) $c_1c_2c_3$_Thresh_Lab220



(c) $c_1c_2c_3$_Forgnd_Lab220

(d) $c_1c_2c_3$_Backgnd_Lab220

(e) $l_1l_2l_3$_Thresh_Lab220



(f) $l_1l_2l_3$_Forgnd_Lab220

(g) $l_1l_2l_3$_Backgnd_Lab220

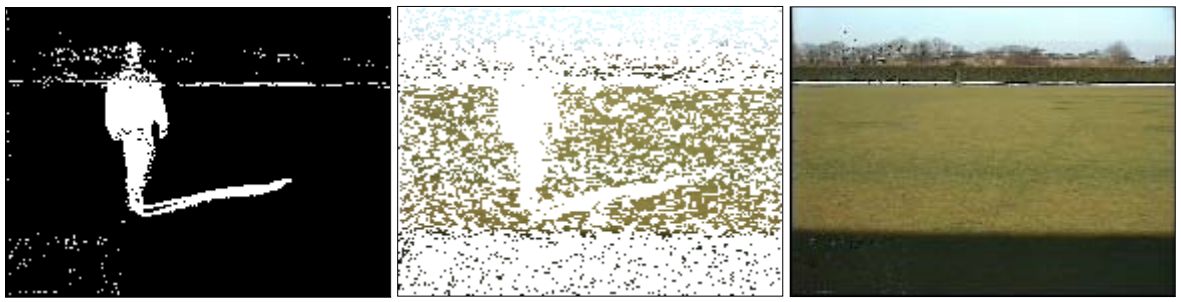
(h) *Nrgb*_Thresh_Lab220 (i) *Nrgb*_Forgnd_Lab220 (j) *Nrgb*_Backgnd_Lab220

(k) *RGB*_Thresh_Lab220 (l) *RGB*_Forgnd_Lab220 (m) *RGB*_Backgnd_Lab220

Fig. 5.7 – (a) The input image Lab220; The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' to Lab220 using $c_1c_2c_3$ ((b)-(d)), $l_1l_2l_3$ ((e)-(g)), normalised *rgb* (*Nrgb*) ((h)-(j)), colour models and the *RGB* colour space ((k)-(m)).

- Sample frames of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Averaging' (Figs. 5.8 to 5.11):

(a) $c_1c_2c_3$_Thresh_Fld290 (b) $c_1c_2c_3$_Forgnd_Fld290 (c) $c_1c_2c_3$_Backgnd_Fld290



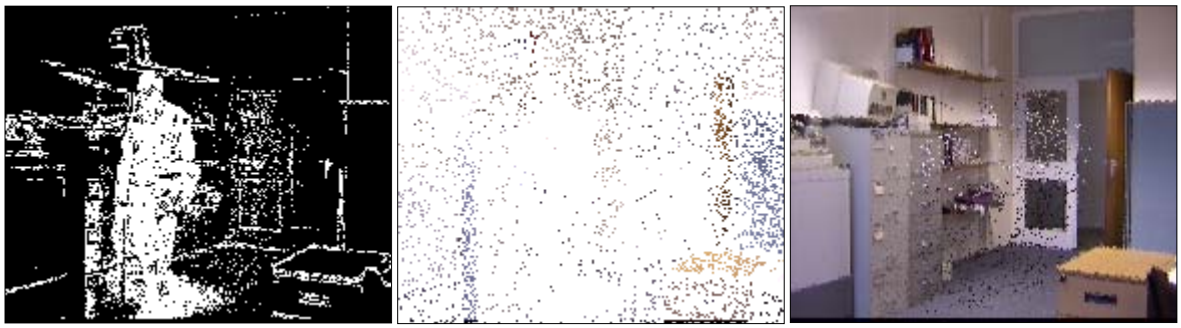
(d) $l_1l_2l_3$_Thresh_Fld290 (e) $l_1l_2l_3$_Forgnd_Fld290 (f) $l_1l_2l_3$_Backgnd_Fld290

(g) *Nrgb*_Thresh_Fld290 (h) *Nrgb*_Forgnd_Fld290 (i) *Nrgb*_Backgnd_Fld290

(j) *RGB*_Thresh_Fld290 (k) *RGB*_Forgnd_Fld290 (l) *RGB*_Backgnd_Fld290

Fig. 5.8 – The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Averaging' to Fld290 using $c_1c_2c_3$ ((a)-(c)), $l_1l_2l_3$ ((d)-(f)), normalised *rgb* (*Nrgb*) ((g)-(i)), colour models and the *RGB* colour space ((j)-(l)).

(a) $c_1c_2c_3$_Thresh_Bijan1_262 (b) $c_1c_2c_3$_Forgnd_Bijan1_262 (c) $c_1c_2c_3$_Backg_Bijan1_262

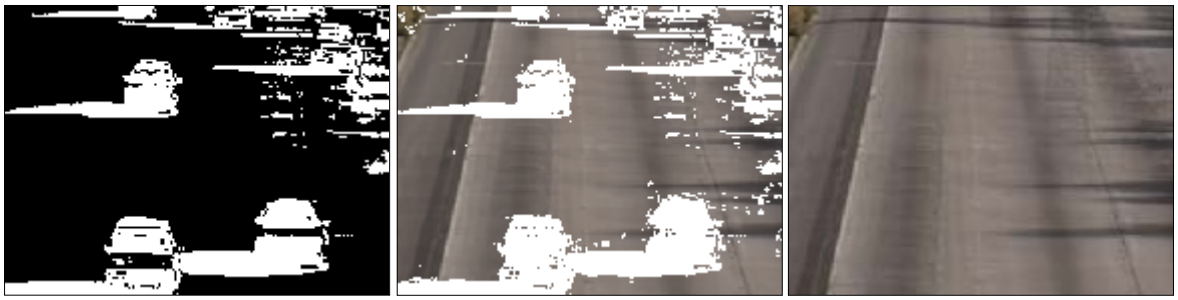


(d) $l_1l_2l_3$_Thresh_Bijan1_262 (e) $l_1l_2l_3$_Forgrnd_Bijan1_262 (f) $l_1l_2l_3$_Backg_Bijan1_262

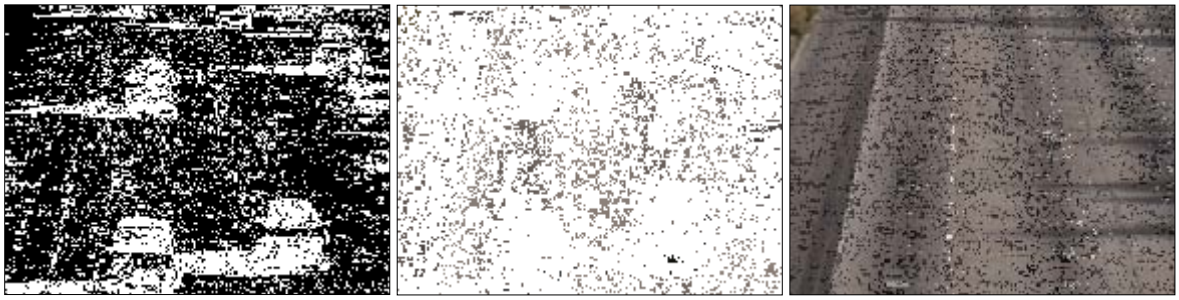
(g) *Nrgb*_Thresh_Bijan1_262 (h) *Nrgb*_Forgrnd_Bijan1_262 (i) *Nrgb*_Backg_Bijan1_262

(j) *RGB*_Thresh_Bijan1_262 (k) *RGB*_Forgrnd_Bijan1_262 (l) *RGB*_Backg_Bijan1_262

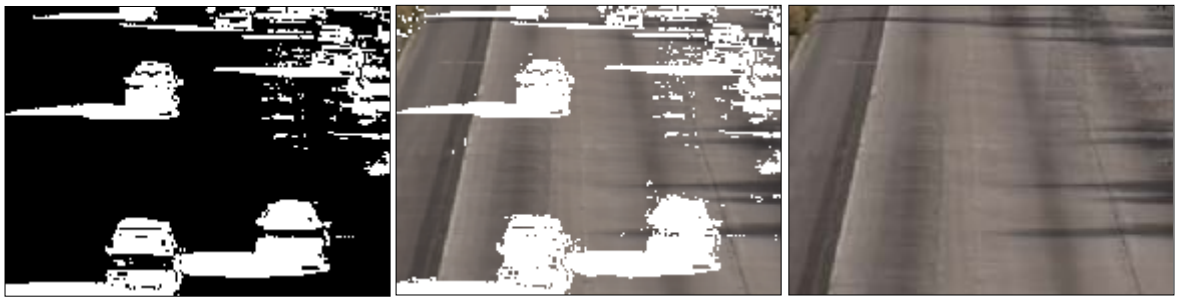
Fig. 5.9 – The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Averaging' to Bijan1_262 using $c_1c_2c_3$ ((a)-(c)), $l_1l_2l_3$ ((d)-(f)), normalised *rgb* (*Nrgb*) ((g)-(i)), colour models and the *RGB* colour space ((j)-(l)).



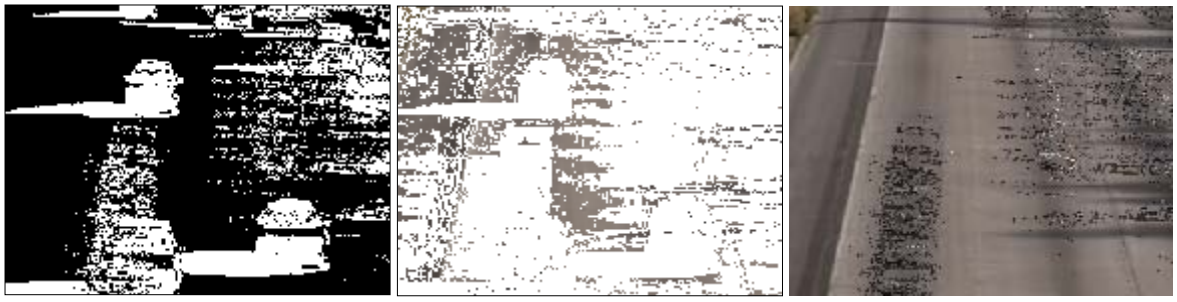
(a) $c_1c_2c_3$_Thrsh_Highway I_261 (b) $c_1c_2c_3$_Forg_Highway I_261 (c) $c_1c_2c_3$_Back_Highway I_261



(d) $l_1l_2l_3$_Thrsh_Highway I_261 (e) $l_1l_2l_3$_Forg_Highway I_261 (f) $l_1l_2l_3$_Back_Highway I_261



(g) *Nrgb*_Thrsh_Highway I_261 (h) *Nrgb*_Forg_Highway I_261 (i) *Nrgb*_Back_Highway I_261



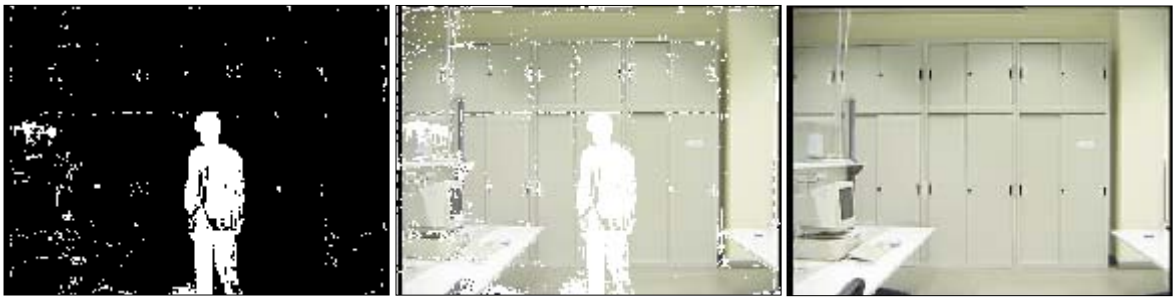
(j) *RGB*_Thrsh_Highway I_261 (k) *RGB*_Forg_Highway I_261 (l) *RGB*_Back_Highway I_261

Fig. 5.10 – The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Averaging' to Highway I_261 using $c_1c_2c_3$ ((a)-(c)), $l_1l_2l_3$ ((d)-(f)), normalised *rgb* (*Nrgb*) ((g)-(i)), colour models and the *RGB* colour space ((j)-(l)).

(a) $c_1c_2c_3$_Thresh_Lab220 (b) $c_1c_2c_3$_Forgnd_Lab220 (c) $c_1c_2c_3$_Backgnd_Lab220

(d) $l_1l_2l_3$_Thresh_Lab220 (e) $l_1l_2l_3$_Forgnd_Lab220 (f) $l_1l_2l_3$_Backgnd_Lab220

(g) *Nrgb*_Thresh_Lab220 (h) *Nrgb*_Forgnd_Lab220 (i) *Nrgb*_Backgnd_Lab220

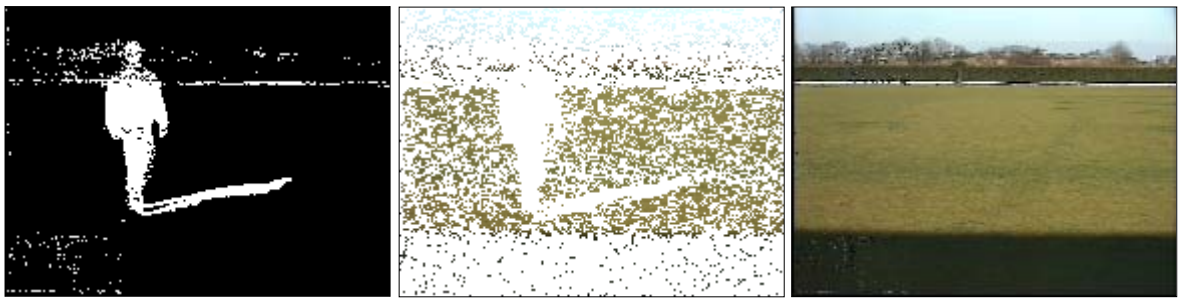


(j) *RGB*_Thresh_Lab220 (k) *RGB*_Forgnd_Lab220 (l) *RGB*_Backgnd_Lab220

Fig. 5.11 – The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Averaging' to Lab220 using $c_1c_2c_3$ ((a)-(c)), $l_1l_2l_3$ ((d)-(f)), normalised *rgb* (*Nrgb*) ((g)-(i)), colour models and the *RGB* colour space ((j)-(l)).

- Sample frames of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Median' (Figs. 5.12 to 5.15):

(a) $c_1c_2c_3$_Thresh_Fld290 (b) $c_1c_2c_3$_Forgnd_Fld290 (c) $c_1c_2c_3$_Backgnd_Fld290



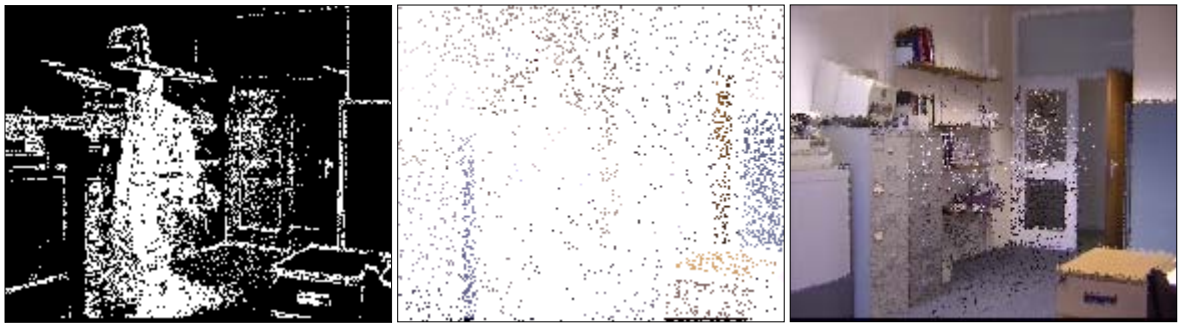
(d) $l_1l_2l_3$_Thresh_Fld290 (e) $l_1l_2l_3$_Forgnd_Fld290 (f) $l_1l_2l_3$_Backgnd_Fld290

(g) *Nrgb*_Thresh_Fld290 (h) *Nrgb*_Forgnd_Fld290 (i) *Nrgb*_Backgnd_Fld290

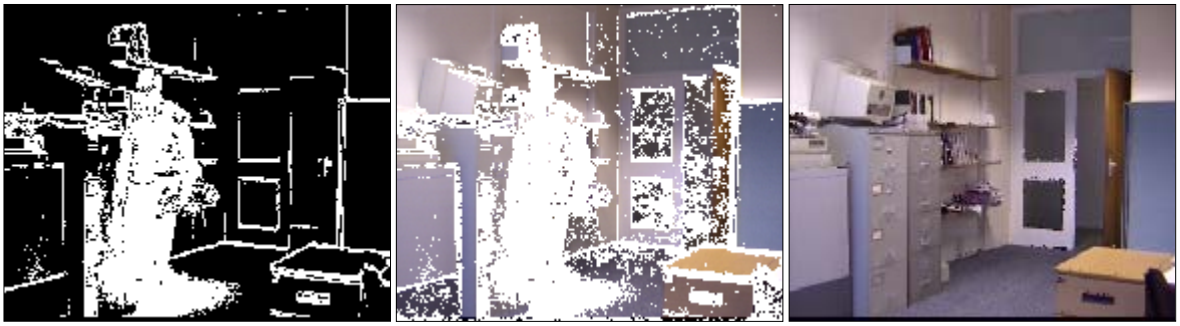
(j) *RGB*_Thresh_Fld290 (k) *RGB*_Forgnd_Fld290 (l) *RGB*_Backgnd_Fld290

Fig. 5.12 – The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Median' to Fld290 using $c_1c_2c_3$ ((a)-(c)), $l_1l_2l_3$ ((d)-(f)), normalised *rgb* (*Nrgb*) ((g)-(i)), colour models and the *RGB* colour space ((j)-(l)).

(a) $c_1c_2c_3$_Thresh_Bijan1_262 (b) $c_1c_2c_3$_Forgnd_Bijan1_262 (c) $c_1c_2c_3$_Backg_Bijan1_262



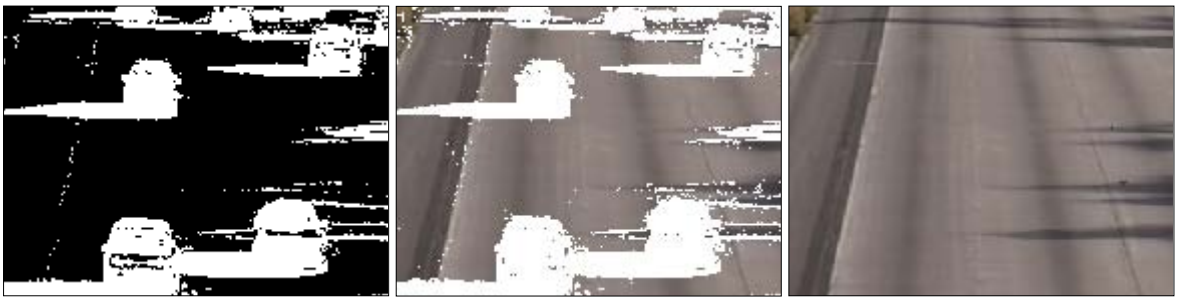
(d) $l_1l_2l_3$_Thresh_Bijan1_262 (e) $l_1l_2l_3$_Forgrnd_Bijan1_262 (f) $l_1l_2l_3$_Backg_Bijan1_262



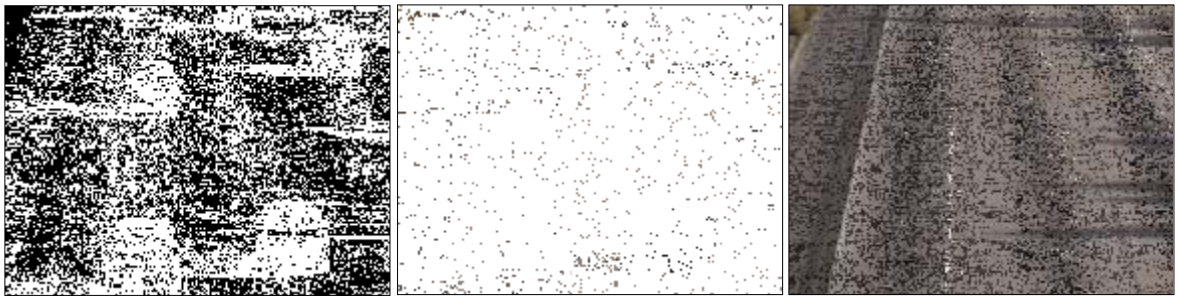
(g) *Nrgb*_Thresh_Bijan1_262 (h) *Nrgb*_Forgrnd_Bijan1_262 (i) *Nrgb*_Backg_Bijan1_262

(j) *RGB*_Thresh_Bijan1_262 (k) *RGB*_Forgrnd_Bijan1_262 (l) *RGB*_Backg_Bijan1_262

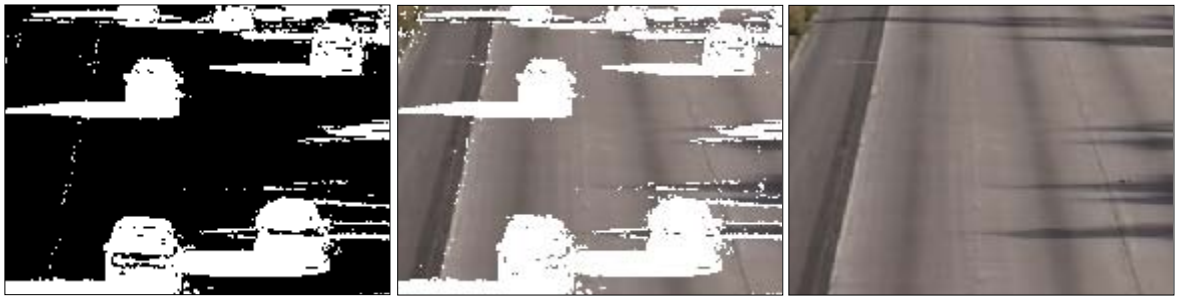
Fig. 5.13 – The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Median' to Bijan1_262 using ((a)-(c)), $l_1l_2l_3$ ((d)-(f)), normalised *rgb* (*Nrgb*) ((g)-(i)), colour models and the *RGB* colour space ((j)-(l)).



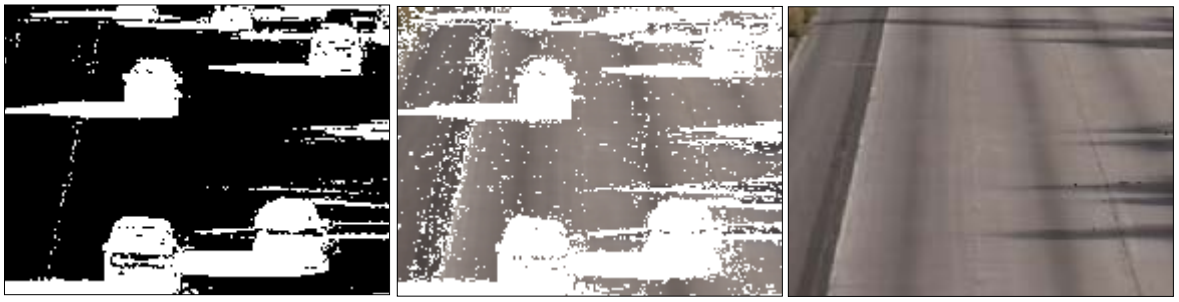
(a) $c_1c_2c_3$_Thrsh_Highway I_261 (b) $c_1c_2c_3$_Forg_Highway I_261 (c) $c_1c_2c_3$_Back_Highway I_261



(d) $l_1l_2l_3$_Thrsh_Highway I_261 (e) $l_1l_2l_3$_Forg_Highway I_261 (f) $l_1l_2l_3$_Back_Highway I_261



(g) *Nrgb*_Thrsh_Highway I_261 (h) *Nrgb*_Forg_Highway I_261 (i) *Nrgb*_Back_Highway I_261



(j) *RGB*_Thrsh_Highway I_261 (k) *RGB*_Forg_Highway I_261 (l) *RGB*_Back_Highway I_261

Fig. 5.14 – The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Median' to Highway I_261 using $c_1c_2c_3$ ((a)-(c)), $l_1l_2l_3$ ((d)-(f)), normalised *rgb* (*Nrgb*) ((g)-(i)), colour models and the *RGB* colour space ((j)-(l)).

(a) $c_1c_2c_3$_Thresh_Lab220 (b) $c_1c_2c_3$_Forgnd_Lab220 (c) $c_1c_2c_3$_Backgnd_Lab220

(d) $l_1l_2l_3$_Thresh_Lab220 (e) $l_1l_2l_3$_Forgnd_Lab220 (f) $l_1l_2l_3$_Backgnd_Lab220

(g) *Nrgb*_Thresh_Lab220 (h) *Nrgb*_Forgnd_Lab220 (i) *Nrgb*_Backgnd_Lab220

(j) *RGB*_Thresh_Lab220 (k) *RGB*_Forgnd_Lab220 (l) *RGB*_Backgnd_Lab220

Fig. 5.15 – The results of applying of the 'Pixel-based' approach based on the 'Selective Update Using Temporal Median' to Lab220 using $c_1c_2c_3$ ((a)-(c)), $l_1l_2l_3$ ((d)-(f)), normalised *rgb* (*Nrgb*) ((g)-(i)), colour models and the *RGB* colour space ((j)-(l)).

The quality figures, which are assigned based on the visual qualities of foreground frames to background images in Figs. 5.4 to 5.15, are offered in Table 5.5. The following conclusions can be made using the Figs. 5.4 to 5.15 and Table 5.5:

- In many foreground frames produced based on $l_1l_2l_3$ colour model, in addition to the pixels of foreground objects, large numbers of background pixels of each input image are often incorrectly identified as foreground pixels (e.g. see 5.4(f), and 5.5(f), 5.6(f), 5.9(e), 5.10(e), 5.11(e), 5.13(e), 5.14(e), and 5.15(e)). As a result, their corresponding background frames are not precise since many pixels are copied from their previous background frames. Besides, sometimes large residues of foreground regions are visible in background images (e.g. see 5.9(f), 5.10(f), 5.11(f), 5.13(f), 5.14(f), and 5.15(f)). Obviously these residues degrade the qualities of background frames which make them unusable. Thus, due to producing inaccurate or low quality background frames, $l_1l_2l_3$ colour model was omitted from Table 5.5.

- The qualities of foreground frames based on $c_1c_2c_3$ and *Nrgb* colour models are very similar to each other. In addition, a lower percentage of background pixels are misclassified as foreground pixels by $c_1c_2c_3$ and *Nrgb* colour models than by the *RGB* colour space (e.g. see 5.4(c), 5.4(i), 5.4(l), 5.6(c), 5.6(i), 5.6(l), 5.10(b), 5.10(h), and 5.10(k)). Thus, $c_1c_2c_3$ and *Nrgb* background frames are often more precise and have better qualities than their correponding *RGB* background frames.

- The 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' produce foreground frames with almost the same qualities as produced by the 'Selective Update Using Non-Foreground Pixels of the Input Image'. However, sometimes fewer numbers of background pixels are incorrectly identified as foreground pixels by the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' than by the 'Selective Update Using Non-Foreground Pixels of the Input Image' (e.g. see 5.7(i), 5.11(h), and 5.15(h)). Thus, a little bit more precise background frames are produced by the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' than by the 'Selective Update Using Non-Foreground Pixels of the Input Image'.

- The 'Selective Update Using Temporal Averaging' utilising *Nrgb* or $c_1c_2c_3$ colour models produces more precise foreground frames for *Fld*, *Bijan1*, and *Lab* video sequences than other selective update methods and colour models in Table 5.5. As a result, it produces background frames with better qualities for *Fld*, *Bijan1*, and *Lab* video sequences.

- The 'Selective Update Using Temporal Median' utilising *Nrgb* or $c_1c_2c_3$ colour models produces more precise foreground frames for *Highway I* video sequence than other selective update methods and colour models in Table 5.5. Thus, it produces better quality background frames for *Highway I* image sequence.

In total, based on Figs. 5.4 to 5.15 and Table 5.5, the qualities of background frames produced by 'Pixel-based' approach are not satisfactorily acceptable. This is because background frames are either not precise enough or residues of foreground regions, although sometimes very slightly (e.g. see 5.7(i), 5.11(h), and 5.15(h)), are visible in them.

The Name of Video Sequence	**The 'Selective Update using Non-Foreground Pixels of the Input Image'**			**The 'Selective Update using Temporal Averaging'**			**The 'Selective Update using Temporal Median'**		
	$c_1c_2c_3$	***Nrgb***	***RGB***	***$c_1c_2c_3$***	***Nrgb***	***RGB***	***$c_1c_2c_3$***	***Nrgb***	***RGB***
Fld	65	68	67	80	82	57	80	81	55
Bijan1	60	63	58	68	73	55	67	69	55
Highway I	57	60	62	85	85	65	88	89	80
Lab	75	77	72	80	80	75	78	78	73

Table 5.5 – The quality figures assigned based on the visual qualities of foreground frames to background images in Figs. 5.4 to 5.15.

5.9 The advantages and disadvantages of the 'Pixel-based' approach

Based on the pseudo code in Fig. 5.3, the 'Pixel-based' approach has the following advantages:

- The 'Pixel-based' approach based on the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' utilising normalised *rgb* is capable to compute the background frame with almost good quality. In addition, it is able to remove the pixels of ghosts after LT frames from reference images (see Fig. 5.16). Meanwhile, it can enclose those pixels of moving foreground objects in the background image that have remained stationary after LT frames (see Fig. 5.17).

The drawbacks of the 'Pixel-based' approach are as follows:

- Many background pixels are excluded from the reference frame by applying a colour filter. Are all excluded pixels really foreground pixels? Can choosing correct values for the thresholds of the colour filter solve all weaknesses of the 'Pixel-based' approach?

- Whatever value is selected for LT (i.e. low or high), two main problems may occur. Since not all the pixel timers of a ghost have the value of LT at the same time, some pixels of background images are replaced sooner than other ones. Such pixels are substituted with the corresponding pixels in input frames. This situation may occur as long as there are some overlaps between the ghost and the current position of the object. As a result, the pixels of a ghost are not removed at the same time but rather over a number of frames. Thus, the resulting background images in which a number of pixels of objects are visible will be invalid for the corresponding input frames (see Fig. 5.16).

- A similar problem occurs when an object becomes motionless. Depending on the timers of all the pixels of an object have the value of LT or not, some pixels (or parts) of the object may be visible in the background frames (see Fig. 5.17). One way to overcome the above problems is to increase LT. However, if LT is increased (e.g. from 50 to 100), the system response time is also increased so that it takes more numbers of frames for a ghost to be removed from the background (see Fig. 5.18). In this case, for more numbers of frames, background images containing ghosts may not represent precise background frames. In fact, this situation may occur for a number of frames in which there are no overlaps between ghosts and the current positions of objects. In a similar manner, a motionless object needs more numbers of frames to be stationary to be included in the background image.

- As stated in the pseudo code in Fig. 5.3, the first input image is regarded as the first background image. Consider the case that the 'Pixel-based' approach produces the first background image where its corresponding input image contains moving foreground objects. The algorithm considers moving foregr-ound objects in the first background image as ghosts. Although the algorithm

can remove them after LT frames, including foreground objects in the reference frame, which should not occur, is regarded as a weakness. Because the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' use a long buffer (more than LT) for updating background pixels, they can usually remove the pixels of ghosts such that no ghost is visible in the first background image. Thus, these methods may not encounter such problem (see Fig. 5.12). On the other hand, if moving objects exist in the first background frame, the 'Selective Update Using Non-Foreground Pixels of the Input Image', which does not use any buffer, can remove the pixels of ghost after LT frames. However, if the pixels of moving objects or their cast shadows overlap with the pixels of ghosts in succeeding frames, the pixels of ghosts may be removed from background frames after a large number of frames (see Fig. 5.12). Thus, in both cases, invalid background frames are produced by the 'Selective Update Using Non-Foreground Pixels of the Input Image'.

In total, based on the quantitative and qualitative results in section 5.8.1 and 5.8.2 and above advantages and disadvantages, the 'Pixel-based' approach as a 'selective update' algorithm, has not a satisfactory performance. Although the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' algorithms have slightly better performances than the 'Selective Update Using Non-Foreground Pixels of the Input Image', the 'Pixel-based' approach has fundamental weaknesses, which are not removed by former algorithms. These disadvantages, which make the 'Pixel-based' approach an ineffective method, are due to the fact that this approach is pixel based. Therefore, a more effective algorithm should be sought.

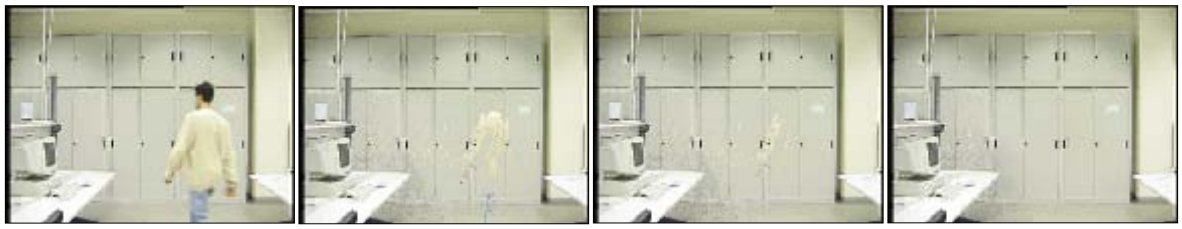
(a) Input image Fld223 (b) Input image Fld272 (c) Input image Fld290 (d) Input image Fld320

(e) Fld_Backgnd223 (f) Fld_Backgnd272 (g) Fld_Backgnd290 (h) Fld_Backgnd320

(i) Inp img Bijan1_200 (j) Inp img Bijan1_249 (k) Inp img Bijan1_260 (l) Inp img Bijan1_275

(m)Bijan1_Backgnd200 (n)Bijan1_Backgnd249 (o)Bijan1_Backgnd260 (p)Bijan1_Backgnd275

(q) Input image Lab215 (r) Input image Lab264 (s) Input image Lab267 (t) Input image Lab280



(u) Lab_Backgnd215 (v) Lab_Backgnd264 (w) Lab_Backgnd267 (x) Lab_Backgnd280

Fig. 5.16 – (a) to (d), (i) to (l), and (q) to (t) are Fld, Bijan1, and Lab input images, respectively. Besides, their corresponding background images are (e) to (h), (m) to (p), and (u) to (x), respectively. The pixels of ghost(s) (i.e. foreground object(s) in the first input image) are removed after LT=50 frames. It takes a number of frames in which ghost(s) is/are completely replaced by the pixels of input images.

(a) Input image Fld400 (b) Input image Fld420 (c) Input image Fld425

(d) Fld_Backgnd400 (e) Fld_Backgnd420 (f) Fld_Backgnd425

Fig. 5.17 – (a) to (c) Fld input images, (d) to (f) corresponding background images, respectively. The moving foreground object has not become stationary but is almost at the same position for many numbers of frames. Because the 'Pixel-based' approach treats the pixels of an object independently and many pixel timers of the object may have the value of LT, those pixels of the object may be visible in the corresponding background image.

(a) Input image Fld223 (b) Input image Fld322 (c) Input image Fld340 (d) Input image Fld355

(e) Fld_Backgnd223 (f) Fld_Backgnd322 (g) Fld_Backgnd340 (h) Fld_Backgnd355

(i) Inp img Bijan1_200 (j) Inp img Bijan1_299 (k) Inp img Bijan1_310 (l) Inp img Bijan1_330

(m)Bijan1_Backgnd200 (n)Bijan1_Backgnd299 (o)Bijan1_Backgnd310 (p)Bijan1_Backgnd330

(q) Input image Lab215 (r) Input image Lab314 (s) Input image Lab317 (t) Input image Lab330

(u) Lab_Backgnd215 (v) Lab_Backgnd314 (w) Lab_Backgnd317 (x) Lab_Backgnd330

Fig. 5.18 – (a) to (d), (i) to (l), and (q) to (t) are Fld, Bijan1, and Lab input images, respectively. Besides, their corresponding background images are (e) to (h), (m) to (p), and (u) to (x), respectively. The pixels of ghost(s) (i.e. foreground object(s) in the first input image) are removed after LT=100 frames. As a result, for many numbers of frames (i.e. LT), ghost(s) is/are visible in the background frames until finally, in a number of frames, they are completely replaced by the pixels of input images.

Chapter 6

An Object-based Approach to Adaptive Dynamic Background Subtraction

6.1 Introduction

In the 'Object-based' approach (Shoushtarian and Bez, 2005), regions of foreground pixels are considered. In this regard, a combination of a size filter and a colour filter distinguishes real foreground objects and adds the remaining regions to the background image. Then, the total initial and returned background pixels are updated using 'Selective Update' methods.

6.2 Introducing the colour filter of the 'Object-based' approach

The pixels in *Foreground-Grp*, *Stopped-Moving-Grp* and *Start-Moving-Grp* plus the exceptions of *Variable-lighting assumption* (i.e. the pixels with relatively large illumination changes) of each input frame usually differ greatly from the corresponding background image. By considering this point, the absolute difference between the current input frame and the previous background image is computed. Then, the appropriate threshold levels for three colour-channels of the difference image are automatically calculated using unimodal thresholding (Rosin, 2001). If for any pixel in the difference image, two of three colour-channels are higher than their corresponding threshold levels, that pixel is marked as a foreground pixel. The remaining pixels are regarded as background pixels (i.e. *Background-Grp*).

After thresholding, a connected components labelling technique is applied to the thresholded difference image (Haralick and Shapiro, 1992). Then, a size filter marks all regions with sizes less than 10 pixels as non-foreground regions (it is assumed that foreground objects are larger than 10 pixels). The pixels of such small regions are added to the background pixels. In addition, the remaining regions are sorted in descending order based on their areas and are assigned area indices (called *region-area-index*) accordingly. In this case, suppose there are *n* remaining regions. Thus,

the *region-area-indices* of the largest, the second largest ……. up to the smallest region will be given the values 0, 1, ….., $n - 1$, respectively.

Next, the pixels of the remaining regions are examined to check whether those regions are really foreground or background ones. A combination of a second size filter and a colour filter is used in this regard. The role of the first size filter is to return very small regions to the background. On the other hand, the second size filter prevents large foreground regions with the lowest *region-area-indices* to be mistakenly marked for deletion by the colour filter. The second size filter compares the *region-area-index* of each region with *Region-Area-Threshold*, which is specified using a table based on the number of the remaining regions (see Table 6.1 which specifies *Region-Area-Threshold*). It is assumed that the largest regions with *region-area-indices* less than *Region-Area-Threshold* are automatically foreground regions. Thus, they don't need to be examined by the colour filter. The reason for this is that thresholding of the difference image behaves well as an intensity filter especially for the largest regions.

No. of Foreground Regions	***Region-Area-Threshold***
1-10	1
11-30	2
31-50	3
> 50	4

Table 6.1 – *Region-Area-Threshold* is specified based on the number of remaining foreground regions after the first size filter.

Based on the results offered in section 5.8.1, normalised *rgb* is used for the comparisons of the colour components of the colour filter in this section. This is due to its better quality and faster computation times in comparison with $l_1l_2l_3$ and $c_1c_2c_3$ invariant colour models and the *RGB* colour space.

The colour filter consists of two parts. The first part considers the pixels of the current input frame corresponding to the pixels of the remaining foreground regions (after applying two size filters). Then, the colour components of these pixels in the current input frame are compared with the colour components of corresponding pixels in two previous background frames. For each pixel of every remaining foreground region, six relative variations are computed based on equation Eq. 6.1 as follows:

$$r1 = | r - r^{*} | / r^{*}, \quad g1 = | g - g^{*} | / g^{*}, \quad b1 = | b - b^{*} | / b^{*},$$
$$r2 = | r - r^{**} | / r^{**}, \quad g2 = | g - g^{**} | / g^{**}, \; b2 = | b - b^{**} | / b^{**} \qquad \text{(Eq. 6.1)}$$

where (r, g, b), (r^{*}, g^{*}, b^{*}) and (r^{**}, g^{**}, b^{**}) are normalised *rgb* colour components of the above pixels in the current input frame, the first and the second previous background frames, respectively. These terms are defined as Max-Val (i.e. a defined maximum value) if their denominators are zero. Then, for every pixel *i* of each

remaining foreground region k (k = 1,…, m, where m is the total number of the remaining foreground regions), the following computation is performed:

Total-Relative-Variation of Pixel $i = r1_i + g1_i + b1_i + r2_i + g2_i + b2_i$ (Eq. 6.2)

Since there are six relative variations for each pixel i of a remaining foreground region k, $rgb\text{-}Avg_k$ is obtained as follows:

$$rgb\text{-}Avg_k \leftarrow (\sum_{i=1}^{n_k} \text{Total-Relative-Variation of Pixel } i) / (6 * n_k), \quad \text{(Eq. 6.3)}$$

where n_k = The number of pixels in the foreground region k

$rgb\text{-}Avg_k$ shows the average of the difference of the colour components of the pixels of a foreground region k in the input image and the colour components of the pixels in the corresponding regions in the previous and the second previous background images.

In the second part of the colour filter, the pixels of remaining foreground regions are considered. For each pixel i in such pixels, its total intensities of the pixels in the current input frame (based on Eq. 5.4) and two previous background frames (i.e. R^*, G^*, B^*, and R^{**}, G^{**}, B^{**}) are obtained. Then, $Intensity\text{-}Avg_k$ for a foreground region k is computed based on equations Eq. 6.4 to Eq. 6.7:

$$I^* \leftarrow R^* + G^* + B^* \quad \text{(Eq. 6.4)}$$

$$I^{**} \leftarrow R^{**} + G^{**} + B^{**} \quad \text{(Eq. 6.5)}$$

$$Relative\text{-}Intensity1 \leftarrow | I - I^* | / I^*,$$

$$Relative\text{-}Intensity2 \leftarrow | I - I^{**} | / I^{**} \quad \text{(Eq. 6.6)}$$

$$Intensity\text{-}Avg_k \leftarrow (\sum_{i=1}^{n_k} (Relative\text{-}Intensity1_i + Relative\text{-}Intensity2_i)) / (2 * n_k) \quad \text{(Eq. 6.7)}$$

$Intensity\text{-}Avg_k$ is similar to $rgb\text{-}Avg_k$. However, in this case the average of the relative variations of the intensities (rather than colour components) of a foreground region k in the input image in comparison with the corresponding regions in two previous background frames is obtained.

The pseudo code in Fig. 6.1 determines whether a region is foreground or background:

// *At first it is assumed that a remaining region is background.*

Foreground-Region ← *False* // *i.e. it's a background region.*
Region-Area-Threshold* ← *A value based on the number of foreground regions
// (*see Table 6.1*)

```
// Check whether the assumption is false and the region is foreground.
// The following if statement performs the second size filter operation.
// It checks whether the region-area-index of a region is smaller than
// Region-Area-Threshold. In this case, it is automatically considered
// as a foreground region without checking by the colour filter (i.e. in
// the else-block of the if statement).

if (region-area-index of a region ≥ Region-Area-Threshold)
{
    // Check remaining regions by the colour filter as follows:

    Area-index ← Region-Area / Total-Region-Areas

    if (((rgb-Avg ≥ Low-rgb-Avg) + (Intensity-Avg ≥ Low-Intensity-Avg) +
          (Area-index ≥ High-Area-index) ≥2)
        or
        (((rgb-Avg ≥ High-rgb-Avg) or (Intensity-Avg ≥ High-Intensity-Avg)) and
          (Area-index ≥ Low-Area-index)))

        Foreground-Region ← True
}
else
    Foreground-Region ← True
```

Fig. 6.1 – The pseudo code for a combination of the second size filter and the colour filter

where the values of 0.06, 0.10, 0.14, 0.25, 0.05 and 0.07 have experimentally been selected for the thresholds *Low-rgb-Avg*, *High-rgb-Avg*, *Low-Intensity-Avg*, *High-Intensity-Avg*, *Low-Area-index*, *High-Area-index*, respectively.

The rationale for the above pseudo code is as follows:

If the *region-area-index*, which indicates the size of a region, is less than *Region-Area-Threshold*, it is assumed to be a foreground region as it was already explained in this section. This technique has the advantage that some larger regions are excluded from checking by the colour filter. These regions sometimes constitute a large percentage of the remaining foreground regions. Thus, it speeds up the processes of the colour filter and the background algorithm for each frame. However, the majority of the remaining foreground regions should be examined by the colour filter.

If a remaining foreground region in the input frame is very similar to the corresponding regions in the two previous background frames, based on our experiments on the tolerance of pixels' *rgb* colour components, its *rgb-Avg* should be less than 5%. *Low-rgb-Avg* is a low threshold because the regions in the input frame have different colour components than the corresponding regions in the previous background frames. So 0.06 is an appropriate value for *Low-rgb-Avg*. If *rgb-Avg* of a region in the input frame is higher than *High-rgb-Avg*, then that region has

relatively different colour components than the corresponding regions in the previous background frames. The value of 0.10 has been chosen for *High-rgb-Avg* based on a number of experiments.

There are similar arguments about the difference of the average intensities of the remaining foreground regions in the input image with their corresponding regions in the previous background frames. Thus, the above values have been chosen for *Low-Intensity-Avg* and *High-Intensity-Avg* experimentally.

It is notable that the values of *rgb-Avg* and *Intensity-Avg* have been used for the comparisons of regions. This makes the comparisons independent from the sizes of foreground regions and thus, it can be applied to any video sequence as well.

The colour filter also uses *Area-index*, which is an indication of the area (or implicitly the weight or the importance) of a region among other remaining foreground regions. If a region has an *Area-index* greater than or equal to *Low-Area-index*, it is approximately considered as a big area. However, if its *Area-index* is greater than *High-Area-index*, then that region should be definitely considered as a big region among all remaining foreground regions.

The colour filter will select a remaining foreground region in the input frame as a foreground region if at least one of the following conditions is met:

1. Either its average colour components are different from two previous background frames by the amount of *Low-rgb-Avg*
   or
   its average intensity is different from two previous background frames by the value of *Low- Intensity-Avg*
   or
   its *Area-index* is considerably high in comparison with other remaining foreground regions.

   At least two of the above conditions should simultaneously be true in order that a region to be selected as a final foreground region.

2. Either its average colour components (i.e. *rgb-Avg*) or its average intensity (i.e. *Intensity-Avg*) should be considerably high and at the same time (for both conditions) its area should not be smaller than *Low-Area-index*.

The pixels of rejected foreground regions by the colour filter in the input image are added to the background. Thus, background pixels after thresholding of the difference image, the pixels of marked regions for deletion by the first size filter, and the pixels of rejected foreground regions by the colour filter are selected for background update. Once all these pixels are updated by one of the ‘Selective Update’ methods discussed in later sections, they are included in the background image. The pixels in the input image corresponding to the final foreground regions are excluded from updating. Instead their corresponding pixels in the previous background frame are included in the current background image.

6.3 Classification of foreground regions

Once the final foreground regions are identified, the 'Object-based' approach still needs to distinguish these foreground regions as *Foreground-Grp*, *Stopped-Moving-Grp* or *Start-Moving-Grp*, even though in special situations *Stopped-Moving-Grp* and *Start-Moving-Grp* are treated as background regions. For distinguishing foreground regions, these regions should be tracked in the image sequence. Since many different situations may occur in practice, the status of motion and the relationship between foreground regions should be determined in a general manner. Fortunately, many object tracking algorithms have already been introduced in the literature.

Matching matrices and matching strings were introduced by Fuentes and Velastin (2006) for object tracking. Besides, 'Merging', 'Splitting', 'Entering', 'Leaving' and 'Correspondence' states (Fuentes and Velastin, 2006) are situations that may happen to regions (or blobs) in general. Matching strings, which can specify these states well, is utilised as a powerful tool for the 'Object-based' approach.

The matching matrices algorithm is based on matching the blobs of the current and the previous input frames. Blob *i* matches blob *j* if their bounding boxes overlap with each other. By comparing the overlap between the bounding boxes of foreground blobs of the current and the previous input images, their matching matrices and matching strings are easily computed. In addition to the bounding box, other information such as area, centroid and average colour (Sechidis, et al., 2002) in each input frame are also kept for blob tracking.

Blobs are assumed to have either 'Stationary' or 'Moving' status. However, after merging or splitting states, new blobs may be created for which the moving status has not been specified. In these cases, they have 'Unknown' status.

The state diagram in Fig. 6.2, illustrates the moving status of foreground blobs. Transitions between states are based on different situations that may happen to a blob in the current input frame in comparison with its overlapping blob in the previous input frame (i.e. 'Merging', 'Splitting', 'Correspondence', 'Entering' and 'Leaving'). The labels on the arrows indicate the starting letter of these states. In addition, it is assumed that a state similar to 'Entering' may also occur. This state is called 'Appearing' and occurs when a new blob or object "appears" in the middle of the current frame (i.e. it does not enter from the image borders). In this case, the new blob is not similar to its overlapping blob(s) in the previous frame (i.e. its area and average colours are different such as a ghost).

Based on the state diagram in Fig 6.2, a blob, which has entered or appeared in the input image, has a moving status ('Entering' or 'Appearing'). A moving blob may maintain its motion or may become motionless (both 'correspondence'). A blob, which was moving in the previous input image, may not be visible in the current frame ('Leaving').

A new blob is visible in the current input frame (i.e. in 'Unknown' state), which is due to merging two or several blobs in the previous frame ('Merging') or is due to splitting of a corresponding blob in the previous frame ('Splitting'). Then a blob,

which has been in ‘Unknown’ state, may be moving, motionless or invisible in the current frame.

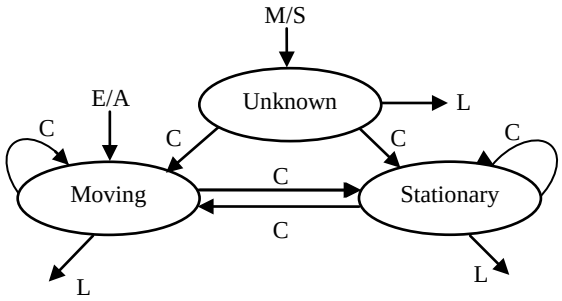


Fig. 6.2 – Initial state diagram for blob (or object) moving status.

A previous stationary blob may remain stationary or it may start moving in the current frame (‘Correspondence’). A stationary object (i.e. a ghost or a completely stationary object) may be replaced by input image pixels after a certain amount of time (‘Leaving’).

By looking at the state diagram, it is observed that ‘Unknown’ and ‘Moving’ states have the same outputs. Thus, in the algorithm, there is not much difference in assuming the blob after merging or splitting to have ‘Unknown’ or ‘Moving’ status as the next states of these two states are the same. Thus, the state ‘Unknown’ can be merged into the state ‘Moving’. Fig. 6.3 illustrates the simplified state diagram. Besides, the resulting moving status of objects is explained in Table 6.2 correspondingly.

In the ‘Stationary’ state in Fig. 6.3, an object may remain motionless for a period of time until it leaves this state. For this purpose, two timers are necessary. These are defined as *st-timer* (stationary timer) and *mv-timer* (moving timer). The *st-timer* counts the number of frames that an object (or a blob) has remained stationary. A blob in frame t is considered stationary if, in comparison with frame $t - 1$, its centroid has changed less than two pixels and its area and average colour have changed by a small amount (e.g. less than 5%). Similarly, the *mv-timer* counts the number of frames that a blob has been in motion. *Ghost-Stationary-Limit* and *Object-Stationary-Limit* are stationary upper limit parameters for *st-timer* and *Object-Moving-Limit* is the moving upper limit parameter for *mv-timer*.

When a background object starts moving, there will be some overlap between its initial position (i.e. ghost) and its current position. Finally, when these two blobs are completely split, the ghost appears as a new blob. The ‘Object-based’ approach does not remove this new blob instantly because it is not completely sure whether this blob is an object or a ghost. So it checks this blob to see whether it remains motionless or it is moving. If it is stationary for *Ghost-Stationary-Limit* frames, then the ‘Object-based’ approach becomes sure that this blob is a ghost and should be replaced by the pixels of the input image. Thus, *Ghost-Stationary-Limit* depends on the system reliability and responsiveness.

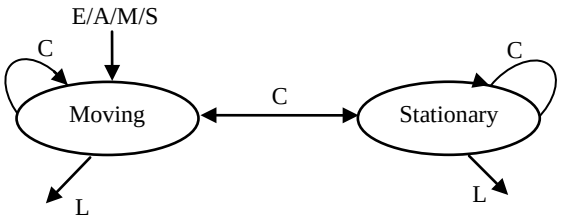


Fig. 6.3 – The simplified state diagram for object moving status

Ghost-Stationary-Limit should be set as minimum as possible. If the bounding box of the ghost overlaps the bounding box of a foreground object before *st-timer* reaches *Ghost-Stationary-Limit*, the ghost will not be replaced by the background pixels of the input image. Instead it is incorrectly detected as a part of a larger foreground object. In fact, *Ghost-Stationary-Limit* should be set such that the 'Object-based' approach is sure that the blob has remained stationary after splitting and the ghost is not occluded again by a foreground object. Thus, it can be set with a minimum number of frames, for example 5 frames.

On the other hand, when an object becomes motionless, the 'Object-based' approach does not consider it as a background object until it remains stationary for a sufficient number of frames (i.e. *Object-Stationary-Limit*). Once again, *Object-Stationary-Limit* depends on the system reliability and responsiveness. In practice, it can be selected from 1 to 2 minutes (1500 to 3000 frames).

	Result
Merging	0 & 1 or 1 & 0 → 1 1 & 1 → 1
Splitting	0 → 1 1 → 1
Correspondence	0 → 0 : (*) 0 → 1 : (+) 1 → 0 : (*)
Entering/Appearing	X → 1
Leaving	0 → : (B), (-) 1 → : (-)

Table 6.2 – This table shows the resultant moving status of a blob in the current frame in comparison with the moving status corresponding blob(s) in the previous frame. Meanwhile, the numbers 0, 1, and X are for: 0: Stationary blob, 1: Moving blob, and X: Unknown-status blob. In addition, the symbols in the parentheses have the following meanings: (*): The parameters (including centroid, size and the average colour) are examined to be close. (+): The parameters are greater than the specified thresholds. (-): The blob information should be removed. (B): The blob is included in the current background image.

In addition to *st-timer* and *mv-timer*, there is another variable for each blob named *st-limit*. *st-limit* is the upper limit variable for *st-timer* and depending on the moving status of each blob; it is initialised with either *Ghost-Stationary-Limit* or *Object-Stationary-Limit*.

In 'Entering' and 'Merging' states, *st-limit* is set with *Object-Stationary-Limit* because the 'Object-based' approach knows that it is concerned with real objects. In 'Splitting' state, *st-limit* is set with *Ghost-Stationary-Limit*. Thus, if a blob moves or is occluded by a foreground object within *Ghost-Stationary-Limit* frames, it will be considered as a foreground object. In other words, if an object interacts with a ghost for less than *Ghost-Stationary-Limit* frames, the ghost will not be replaced with the input image. In 'Appearing' state, *st-limit* is also set with *Ghost-Stationary-Limit*. In 'Correspondence' state, *st-limit* will receive its parent's *st-limit* value.

In all states, if the blob is moving, *mv-timer* and *st-timer* will be set with one and zero, respectively. Only in 'Correspondence' state, if a blob is stationary in both frames t - 1 and t, *st-timer* will be set by its parent's *st-timer* plus one and *mv-timer* will be reset to zero. However, if the blob was moving in the previous frame and is also moving in the current frame, *st-timer* will be reset to zero and *mv-timer* is set with its parent's *mv-timer* plus one. Then, *mv-timer* will also be checked with *Object-Moving-Limit* and if they are equal, the *st-limit* is set with *Object-Stationary-Limit*. Otherwise, *st-limit* will be set with its parent's *st-limit*. The reason *mv-timer* is compared with *Object-Moving-Limit* is that the 'Object-based' approach needs to check that the blob is not a ghost, which was moving in the past few frames. In this case, the 'Object-based' approach should be sure of the movement of a blob before it assigns the blob's *st-limit* with *Object-Stationary-Limit*. Like *Ghost-Stationary-Limit*, *Object-Moving-Limit* can be set with a few numbers of frames such as 5.

After the matching strings S_t^{t-1} and S_{t-1}^{t} are computed (Fuentes and Velastin, 2006), the columns of S_t^{t-1} will be compared with the columns of S_{t-1}^{t}. Then, the moving status of each blob in the current frame is specified based on the moving status of the corresponding blob in the previous frame and the movement change that has currently happened to this blob. Besides, the blob's variables are set accordingly. The result of such a comparison for each blob is to determine the value of a flag called BF-flag (background or foreground flag). For all states, this flag is assigned with 'F'. This means that this blob is considered as a foreground blob and the 'Object-based' approach will replace its pixels with the corresponding pixels of the previous background image. However, only in 'Correspondence' state, if a blob has been stationary in the previous and the current frame, *st-timer* will be checked with *st-limit* and just when they are equal BF-flag is assigned with 'B'. In this case, the pixels of the input image corresponding to the pixels of this blob are copied in the background image.

It is important to notice that if *st-limit* has the value of *Ghost-Stationary-Limit*, then the blob is a ghost and it will be removed from the background frame. But if *st-limit* has the value of *Object-Stationary-Limit*, it is an object that has been stationary for a large number of frames. So it will now be considered a part of the

background image. A summary of the moving status, timers and flags of all states in the 'Object-based' approach is given in Table 6.3.

Merging:	***Entering:***	***Appearing:***
Moving-status ← Moving	*Moving-status* ← Moving	*Moving-status* ← Moving
st-timer ← 0	*st-timer* ← 0	*st-timer* ← 0
mv-timer ← 1	*mv-timer* ← 1	*mv-timer* ← 1
st-limit ← *Object-Stationary -Limit*	*st-limit* ← *Object-Stationary -Limit*	*st-limit* ← *Ghost-Stationary -Limit*
BF-Flag ← 'F'	BF-Flag ← 'F'	BF-Flag ← 'F'

Splitting:
Moving-status ← Moving
st-timer ← 0
mv-timer ← 1
if (parent ='M' or parent ='S' or parent ='E')
  st-limit ← *Object-Stationary-Limit*
else // if (parent = 'A')
  st-limit ← *Ghost-Stationary-Limit*

Correspondence & Changing States:

// *A stationary object in the* // *previous frame is motionless* // *in the current frame*.	// *A stationary object is now* // *moving in the current* // *frame*.	// *A moving object now has* // *no movement in the* // *current frame*.
0 → 0**:**	0 → 1**:**	1 → 0**:**
Moving-status ← Stationary	*Moving-status* ← Moving	*Moving-status* ← Stationary
mv-timer ← 0	*mv-timer* ← 1	*mv-timer* ← 0
st-timer ← parent st-timer + 1	*st-timer* ← 0	*st-timer* ← 1
st-limit ← parent *st-limit*	*st-limit* ← parent *st-limit*	*st-limit* ← parent *st-limit*
if (*st-timer* = parent *st-limit*)	BF-Flag ← 'F'	BF-Flag ← 'F'
BF-Flag ← 'B'		
else		
BF-Flag ← 'F'		

// *A moving object is still moving in the current frame*.
1 → 1**:**
Moving-status ← Moving
st-timer ← 0
mv-timer ← parent *mv-timer* + 1
if (*mv-timer* = *Object-Moving-Limit*)
  st-limit ← *Object-Stationary-Limit*
else
  st-limit ← parent *st-limit*
BF-Flag ← 'F'

Table 6.3 – A summary of the moving status, timers and flags of all states in the 'Object-based' approach. 'B' and 'F' flags mean background and foreground, respectively.

Therefore, by using matching strings, the 'Object-based' approach is able to identify the behaviours of all blobs regardless of their numbers effectively.

6.4 Performances of three ‘Selective Update’ algorithms

In subsection 6.4.1, the performance and the weaknesses of the ‘Selective Update Using Non-Foreground Pixels of the Input Image’ will be explained. The performances and the capabilities of the ‘Selective Update Using Temporal Averaging’ and the ‘Selective Update Using Temporal Median’ for resolving the shortcomings of the ‘Selective Update Using Non-Foreground Pixels of the Input Image’ will be demonstrated in subsections 6.4.2 and 6.4.3, respectively. Then, the results of applying three ‘Selective Update’ algorithms to eight video sequences are illustrated in subsection 6.4.4.

6.4.1 The performance of the ‘Selective Update Using Non-Foreground Pixels of the Input Image’ algorithm

One drawback of the ‘Selective Update Using Non-Foreground Pixels of the Input Image’ is that the pixels in *Background-Grp* of the input image t are not necessarily close to the corresponding pixels in the background image $t - 1$. This situation is more evident in the outdoor environments where pixels often have relatively high lighting fluctuations. This may cause the threshold levels for the difference image to be computed as high levels. Since some objects have soft cast-shadows, a number of soft shadow pixels may be classified in *Background-Grp* due to high threshold levels for the difference image. So the ‘Selective Update Using Non-Foreground Pixels of the Input Image’ algorithm will copy all *Background-Grp* pixels of the input frame in the background frame. Thus, as the objects move in the succeeding input frames, integrations of soft shadow pixels will be visible in some parts of the corresponding background frames (see Fig. 6.4c and 6.4d). Lowering the threshold levels for the difference image can alleviate this problem but can not completely solve it.

As the second drawback, the colour filter of the ‘Object-based’ approach may incorrectly classify a remaining region very rarely (after conntected components labelling) as a background area while it must be classified as a foreground region. For example, the pixels of a region of soft shadow or a combination of a large soft shadow area and a small piece of a foreground object may be very similar to the corresponding pixels of the previous background frames. In these cases, the ‘Object-based’ approach may not classify those areas as foreground regions. One obvious solution for this misidentification is to lower the threshold levels of the colour filter. However, lowering the threshold levels may cause many background areas of each input frame to be similarly misidentified as foreground regions. Thus, the thresholds should be suitably chosen based on various experiments.

Thus, due to the copying feature of the ‘Selective Update Using Non-Foreground Pixels of the Input Image’ algorithm, it may fail to produce a correct background image in the above cases. To resolve these problems, let’s consider the ‘Selective Update Using Temporal Averaging’ and the ‘Selective Update Using Temporal Median’ algorithms.

6.4.2 The performance of the 'Selective Update Using Temporal Averaging' algorithm

The second weakness of the 'Selective Update Using Non-Foreground Pixels of the Input Image' was explained to be occasionally misidentifying a foreground region, which in this case is similar to the background. Hence, giving rise to the question "Will the 'Selective Update Using Temporal Averaging' be able to rectify the 'Selective Update Using Non-Foreground Pixels of the Input Image' shortcomings"?

The 'Selective Update Using Temporal Averaging' adds a weight of $1/M$ of a pixel in *Background-Grp* (for each colour-channel) of the input image to the same weight of $M-1$ corresponding previous background pixels, where $M+1$ is the length of a cyclic buffer which is used for storing the background frames. Thus, as a higher value is chosen for M, the pixels in *Background-Grp* of each input frame have less effect on the background frame. But the 'Selective Update Using Temporal Averaging' cannot completely resolve this effect unless M has a high value. For the following reasons very high values cannot be chosen for M:

- If the value of M is very high, the same weight is given to very old previous background frames as is given to the pixels of *Background-Grp* in the current input image. In this case, the 'Selective Update Using Temporal Averaging' is unable to quickly update and adapt the background image with the latest changes in recent input images. Thus, the obsolete background image may lead to failure of later processes such as detection and tracking.

- As the value of M increases, the volume of the cyclic buffer for the background images increases directly. Thus, an indefinite value for M cannot be selected.

Therefore, the 'Selective Update Using Temporal Averaging' is unable to remove the second weakness of the 'Selective Update Using Non-Foreground Pixels of the Input Image' effectively. In fact, for example, if a small foreground region is missed by the colour filter of the 'Object-based' approach, a blurring effect of the foreground region will still be visible in the background image. However, despite this weakness, if no foreground region is missed by the colour filter of the 'Object-based' approach, then the 'Selective Update Using Temporal Averaging' produces correct background frames in real-time. It also removes the first weakness of the 'Selective Update Using Non-Foreground Pixels of the Input Image' as is explained in subsection 6.4.4.

6.4.3 The performance of the 'Selective Update Using Temporal Median' algorithm

In the 'Selective Update Using Temporal Median', each colour-channel of a pixel in *Background-Grp* of the input image is read into a histogram, which also contains M corresponding colour-channels of pixels of the previous background frames. Consider a foreground region which is mistakenly identified as a background area.

Since the pixels of this region are similar but not exactly the same as the pixels of the previous background frames, there is a very low probability that each pixel of the foreground region is selected as the median. Thus, the shortcoming of the 'Selective Update Using Non-Foreground Pixels of the Input Image' does not affect the 'Selective Update Using Temporal Median' to compute the correct background frame effectively. In fact, to resolve both weaknesses of the 'Selective Update Using Non-Foreground Pixels of the Input Image', the length of the cyclic buffer ($M + 1$) should be relatively long (e.g. more than 50). However, the 'Selective Update Using Temporal Median' must be implemented such that it can operate as fast as possible.

For fast computation of the 'Selective Update Using Temporal Median', a histogram is built for each colour-channel of a pixel in the background image. For this purpose, a number of initial background frames are read into histograms and the median (bin) and the position of median in the median bin (for each colour-channel) are computed and stored.

When the 'Selective Update Using Temporal Median' algorithm operates continuously, a colour-channel of a pixel in *Background-Grp* of the input image is read into the histogram and the corresponding colour-channel of a pixel in the *M*th previous background frame is removed from the histogram. In this case, depending on the relative positions of the values entered and removed from the histogram to the median bin, the median bin and the position of the median can easily be modified. However, for the pixels of the foreground regions of the input image, the corresponding histograms are not changed.

Therefore, using the above technique, the 'Selective Update Using Temporal Median' can be implemented effectively.

6.4.4 The results of three 'Selective Update' algorithms

Due to filtering effect of the 'Selective Update Using Temporal Averaging' and 'Selective Update Using Temporal Median' in comparison with the 'Selective Update Using Non-Foreground Pixels of the Input Image', the pixels in the background frame t are usually closer to the similar pixels in *Background-Grp* of the input image $t + 1$. Hence, more background pixels are correctly identified and foreground regions are better recognised. This advantage of the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' also eases the effectivity of 'connected components labelling', 'colour filter' and other later processes as well. The results of applying three 'Selective Update' algorithms to eight video sequences, 'Lab', 'Fld', 'Cp1', 'Cp3', 'Cp5', 'Highway I', 'Highway II' and 'Bijan1' are illustrated in Figs. 6.4, 6.5 and 6.6. The DR and FAR vaues of three 'Selective Update' algorithms are also given in Table 6.4.

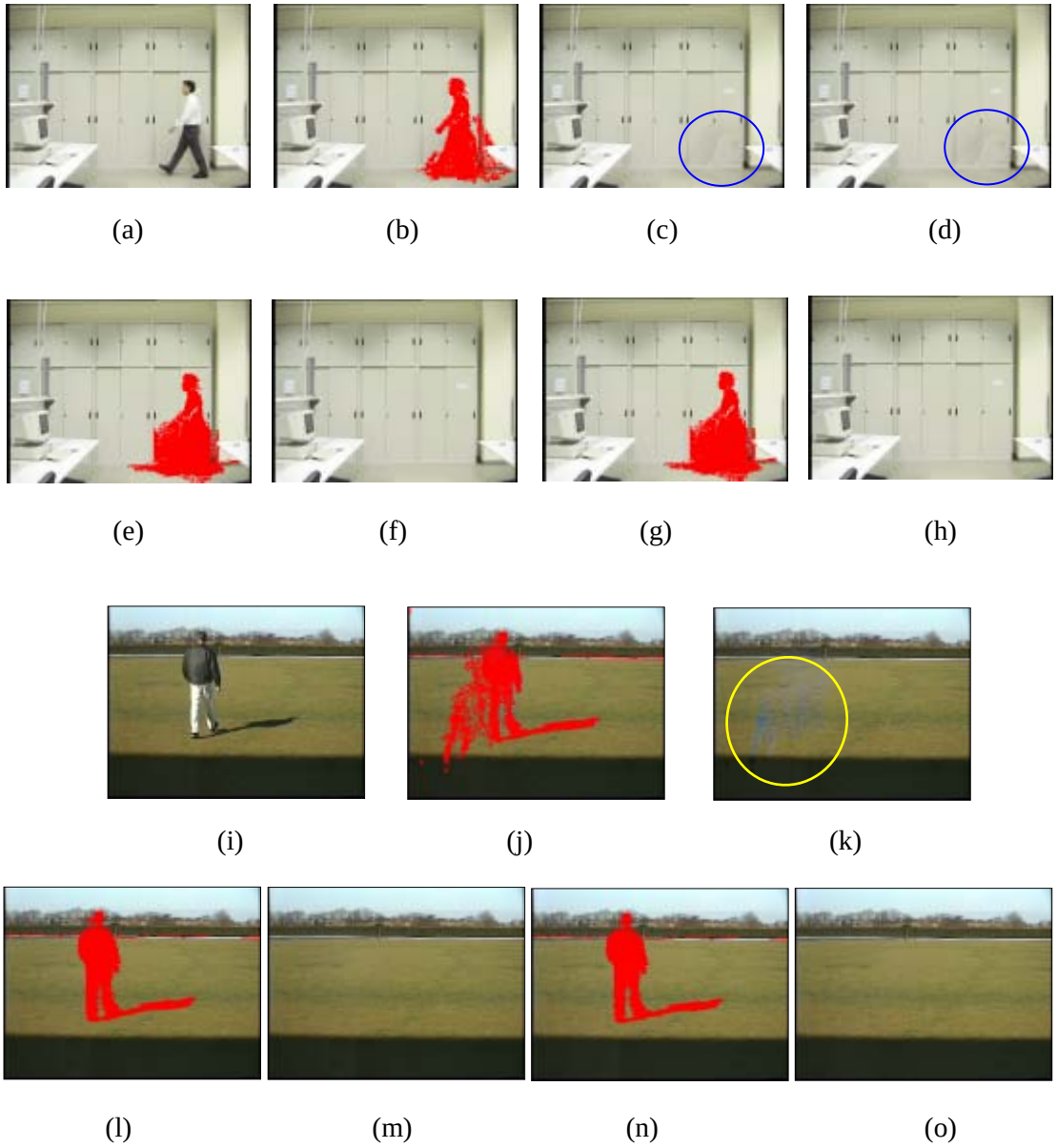


Fig. 6.4 – (a) Input image Lab148, (b) Detected foreground region Non_forg_Lab_Fgnd148, (c) and (d) The previous and the current background images Non_forg_Lab_back147 and Non_forg_Lab_back148, respectively. The Soft shadow pixels are almost visible inside the drawn ovals; (e) Foreground region Temp_Avg_Lab_Fgnd148, (f) Temp_Avg_Lab_back148, (g) Foreground region Temp_Med_Lab_Fgnd148, (h) Temp_Med_Lab_back148. No soft shadow pixels are visible in the background images (f) and (h). (i) Input image Fld298, (j) Foreground region Non_forg_Fld_Fgnd298; (k) Non_forg_Fld_back298. Some residue pixels (or regions) are visible inside the drawn oval. (l) and (n) Detected foreground regions Temp_Avg_Fld_Fgnd298 and Temp_Med_Fld_Fgnd298; (m) and (o) Corresponding background images Temp_Avg_Fld_back298 and Temp_Med_Fld_back298, respectively.

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

(k)

(l)

(m)

(n)

(o)

(p)

(q)

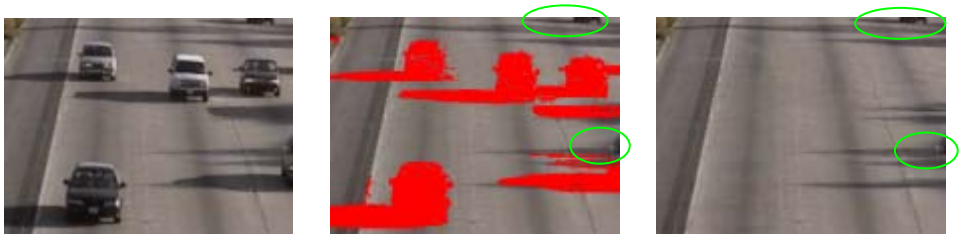
(r)

(s)

(t)

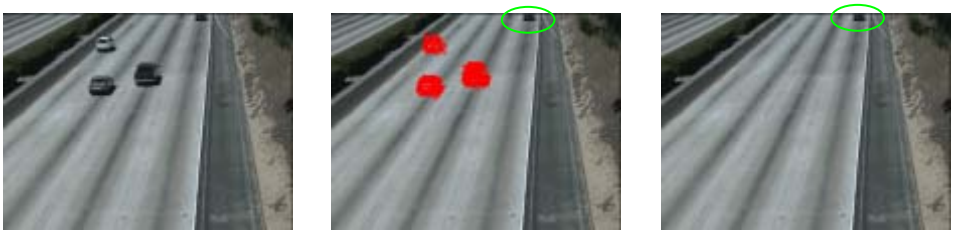
(u)

Fig. 6.5 – (a), (h) and (o), Input images Cp1_381, Cp3_149 and Cp5_139, respectively; (b and c), (i and j) and (p and q) are respectively detected foreground regions and their corresponding background images based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' for the corresponding input images Cp1_381, Cp3_149 and Cp5_139; (d and e), (k and l) and (r and s) are respectively detected foreground regions and their corresponding background images based on the 'Selective Update Using Temporal Averaging' for the mentioned input images; (f and g), (m and n) and (t and u) are respectively detected foreground regions and their corresponding background images based on the 'Selective Update Using Temporal Median' for the mentioned input images.



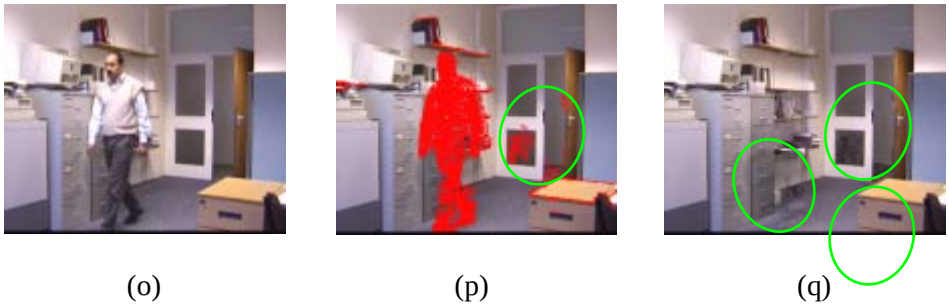
(a) (b) (c)

(d) (e) (f) (g)



(h) (i) (j)

(k) (l) (m) (n)



(o) (p) (q)

(r) (s) (t) (u)

Fig. 6.6 – (a), (h) and (o), Input images Highway I_268, Highway II_126 and Bijan1_260; (b and c), (i and j) and (p and q) are detected foreground regions and their corresponding background images based on the 'Selective Update Using Non-Foreground Pixels of the Input Image' for the corresponding input images Highway I_268, Highway II_126 and Bijan1_260; (d and e), (k and l) and (r and s) are detected foreground regions and their corresponding background images based on the 'Selective Update Using Temporal Averaging' for the mentioned input images; (f and g), (m and n) and (t and u) are detected foreground regions and their corresponding background images based on the 'Selective Update Using Temporal Median' for the mentioned input images. Some foreground objects and residue regions (including shadow pixels) are visible in (c), (j) and (q) background images. They indicate the failure of the 'Selective Update Using Non-Foreground Pixels of the Input Image' algorithm in some cases. However, the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median' have very good performances in all these cases as no foreground objects or residue regions are seen in their corresponding background images.

The average computation times of three 'Selective Update' algorithms on the above video sequences are given in Table 6.5. Three 'Selective Update' algorithms have been implemented on a 2.4 GHz Pentium 4 PC with 1 GB of RAM running windows XP professional edition. From Table 6.5, it is obvious that both the 'Selective Update Using Non-Foreground Pixels of the Input Image' and the 'Selective Update Using Temporal Averaging' run in real-time (less than 40 mSec) for eight image sequences. However, the 'Selective Update Using Temporal Median' runs close to real-time for standard images (with resolution 320 x 240, e.g. 'Lab', 'Highway I' and 'Highway II'). Fortunately, for faster PCs (with higher CPUs' and mother boards' speeds), the 'Selective Update Using Temporal Median' easily runs in real-time even for image sequences with relatively high resolutions (e.g. 384 x 288).

6.4.5 The advantages of the 'Selective Update Using Temporal Median' method

It was shown in the previous sections that the 'Selective Update Using Temporal Median' has the best performance among the three 'Object-based Selective Update' algorithms. Its advantages are as follows:

- It produces a dynamic reference image for each input frame.

- The proposed algorithm is very simple and easy to understand and implement.

Image Sequence		Selective Update Using Non-Foreground Pixels of the Input Image	Selective Update Using Temporal Averaging	Selective Update Using Temporal Median
Lab	**DR**	**0.976**	**0.986**	**0.994**
	FAR	**0.192**	**0.032**	**0.017**
Fld	**DR**	**0.961**	**0.995**	**0.996**
	FAR	**0.067**	**0.010**	**0.009**
Cp1	**DR**	**0.769**	**0.975**	**0.975**
	FAR	**0.001**	**0.007**	**0.004**
Cp3	**DR**	**0.780**	**0.989**	**0.988**
	FAR	**0.015**	**0.024**	**0.021**
Cp5	**DR**	**0.982**	**0.986**	**0.991**
	FAR	**0.137**	**0.046**	**0.021**
Highway I	**DR**	**0.968**	**0.983**	**0.985**
	FAR	**0.003**	**0.001**	**0.001**
Highway II	**DR**	**0.960**	**0.985**	**0.985**
	FAR	**0.013**	**0.005**	**0.002**
Bijan1	**DR**	**0.944**	**0.974**	**0.975**
	FAR	**0.008**	**0.004**	**0.003**

Table 6.4 – The DRs and FARs of three 'Selective Update' methods for 'Lab', 'Fld', 'Cp1' , 'Cp3' ,'Cp5', 'Highway I', 'Highway II', and 'Bijan1' video sequences.

- It has been designed to use colour video sequences and is robust since it can operate in unconstraint outdoor or indoor scenes.

- The three parameters for thresholding every difference image are computed automatically. The other parameters of the algorithm are fixed and the same set of parameters is used for different video sequences. Thus, the algorithm does not require that its parameters are set for each video sequence manually.

- It also removes ghosts and includes stopped moving object in the background image in one frame effectively. Meanwhile, the total routines of the algorithm run close to real-time on a 2.4 GHz Pentium 4 for standard image sequences.

6.5 Comparison with related works

In section 6.5.1, the criteria of comparison of background generation algorithms are offered. Then, 'Object-based Selective Update using Temporal Median' is compared with 'Pixel-based Selective Update using Temporal Averaging' and 'Pixel-based Selective Update Using Temporal Median' and two other background generation algorithms in sections 6.5.2 and 6.5.3, respectively.

Image Sequence (resolutions in parentheses)	**The 'Selective Update Using Non-Foreground Pixels of the Input Image'**	**The 'Selective Update Using Temporal Averaging'**	**The 'Selective Update Using Temporal Median'**
Lab **(320 x 240)**	**7.0**	**14.6**	**47.4**
Fld **(384 x 288)**	**8.7**	**18.8**	**64.3**
Cp1 **(352 x 288)**	**9.7**	**17.5**	**63.0**
Cp3 **(352 x 288)**	**9.2**	**17.6**	**61.3**
Cp5 **(352 x 288)**	**8.4**	**17.8**	**61.7**
Highway I **(320 x 240)**	**6.9**	**14.2**	**42.2**
Highway II **(320 x 240)**	**7.2**	**14.8**	**47.9**
Bijan1 **(352 x 288)**	**8.0**	**18.0**	**60.2**

Table 6.5 – The average computation times for three 'Selective Update' algorithms in mSec. It is notable that moving objects in 'Cp1', 'Cp3' and 'Cp5' image sequences constitute a very small percentage of each input frame in comparison with 'Fld' video sequence. Thus, often more than 95% of the pixels are classified as background pixels which should be updated using the corresponding 'Selective Update' method. This is the reason that the computation times of 'Cp1', 'Cp3' and 'Cp5' are relatively high in comparison with 'Fld' sequence which has a higher resolution.

6.5.1 The criteria of comparison

Good performance, correctness and the speed of computations are the criteria of comparisons in this section. The concept of good performance is defined as follows:

- It can correctly operate in both outdoor and indoor scenes continuously for any period of time.

- The pixels of foreground objects are not detected as background pixels and vice versa. In Cucchira et al. (2001) this definition was expressed by two terms 'Detection Rate – DR' and 'False Alarm Rate – FAR' as follows:

  DR = TP/(TP+FN) and FAR = FP/(TP+FP), where TP is the number of the pixels of correctly detected moving object, FN is the pixels of the missed moving object, and FP is the background pixels incorrectly detected as the pixels of the moving object. Based on the contents of input images, the algorithm's parameters are computed automatically. In other words, the algorithm's parameters should not be chosen (or tuned) for each video sequence manually.

The concepts of correctness are defined as follows:

- To identify ghosts and to be able of removing their pixels at the same time in one frame.

- To identify moving objects which have become stationary for a large number of frames. Also to be able to include the pixels of a stationary object in the background image at the same time in one frame and not over a number of frames.

The other important criterion of comparison is the speed of computations as follows:

- It is desirable and important that the algorithm can finish all its routines in real-time.

6.5.2 Comparison of the 'Object-based' approach and the 'Pixel-based' approach

It was shown in Chapter 4 that the 'Selective Update Using Temporal Averaging' and the 'Selective Update using Temporal Median' utilising normalised *rgb* were the most suitable methods for implementing the 'Pixel-based' approach. Of course, their performances were not sufficiently satisfactory. On the other hand, it was shown in Section 6.4.3 that the 'Selective Update using Temporal Median' has the best performance among three 'Selective Update' methods of the 'Object-based' approach. In this section, the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' methods are compared with the 'Object-based Selective Update using Temporal Median' algorithm. They are compared quantitatively, qualitatively, and also based on the criteria of comparison given in Section 6.5.1.

- Based on quantitative comparison:

For quantitative comparison, the 'Pixel-based Selective Update using Temporal Averaging', the 'Pixel-based Selective Update using Temporal Median' and the 'Object-based Selective Update using Temporal Median' methods are applied to four video sequences *Fld*, *Bijan1*, *Highway I*, and *Lab* based on normalised *rgb*. Then, for corresponding numbers of frames of these video sequences, the details of quantitative comparisons of the above methods based on DR and FAR values are offered in Table 6.6.

It is observed from Table 6.6 that the DRs of the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' are as high as the DRs of the 'Object-based Selective Update using Temporal Median'. But the FARs of the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' are much higher than the FARs of the 'Object-based Selective Update using Temporal Median' method. The reason for this is as follows:

Unimodal thresholding is used for thresholding colour difference images of both the 'Pixel-based' and the 'Object-based' methods. As explained in Section 4.7 about the 'Pixel-based Selective Update' methods, in addition to the pixels of foreground objects in thresholded images, a large number of background pixels are incorrectly classified as foreground pixels by the colour filter. In this case,

Image Sequence		**The 'Pixel-based Selective Update Using Temporal Averaging'**	**The 'Pixel-based Selective Update Using Temporal Median'**	**The 'Object-based Selective Update Using Temporal Median'**
Fld	**DR**	**0.996**	**0.996**	**0.996**
	FAR	**0.591**	**0.618**	**0.009**
Bijan1	**DR**	**0.978**	**0.975**	**0.975**
	FAR	**0.262**	**0.317**	**0.003**
Highway I	**DR**	**0.988**	**0.989**	**0.985**
	FAR	**0.032**	**0.040**	**0.001**
Lab	**DR**	**0.977**	**0.975**	**0.994**
	FAR	**0.163**	**0.189**	**0.017**

Table 6.6 - Quantitative comparison of the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' with the 'Object-based Selective Update using Temporal Median' based on detection rate (DR) and false alarm rate (FAR) values.

almost all pixels of real foreground objects are identified as foreground pixels. Thus, the 'Pixel-based Selective Update' methods have very high DRs. On the other hand, misclassifying a large number of background pixels as foreground pixels causes these methods also have very high FARs. As a result, the 'Pixel-based Selective Update' methods do not produce precise background frames.

In the 'Object-based Selective Update using Temporal Median' method, after connected components labelling process, pixels of very small foreground regions are removed and added to background pixels by a size filter. In addition, a colour filter distinguishes real foreground objects and adds the pixels of remaining regions to the background frame. For example, after thresholding, small background regions with slight movements are sometimes classified as foreground regions. But the colour filter may reject them due to their similarities with corresponding regions in previous background frames. In this way, real foreground regions are identified. Meanwhile, a few background pixels may misclassify as foreground pixels. Thus, the 'Object-based Selective Update using Temporal Median' has very high DRs and very low FARs (see Table 6.6). Therefore, it produces high precision background frames.

- Based on qualitative comparison:

The sample frames produced by applying the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' to four video sequences *Fld*, *Bijan1*, *Highway I*, and *Lab* based on normalised *rgb* were already shown in Figs. 5.8 to 5.15. In addition, the quality figures of these methods were also given in Table 5.5. Corresponding thresholded, foreground, and background frames due to applying the 'Object-based Selective Update using Temporal Median' to above video sequences are shown in Fig. 6.7. Similar to Table 5.5, the quality figures assigned based on the visual qualities of foreground frames to background images in Fig. 6.7 for the 'Object-based Selective Update using Temporal Median' are given in Table 6.7. For qualitative comparison, Table 6.7 includes the quality figures of the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' from Table 5.5 as well.

By comparing the quality figures of Table 6.7 and Fig. 6.7 with corresponding Figs. 5.8 to 5.15 the following conclusion can me made:

The 'Object-based Selective Update using Temporal Median' detects foreground objects accurately while it may rarely miss few small foreground regions. In addition, almost all reference pixels are correctly identified as background pixels (Fig. 6.7 (b), (e), (h), and (k)). Thus, the reference pixels of each input image are considered in 'Temporal Median' process. As a result, precise, up to date, and very good quality background fames are computed for input images (Fig. 6.7 (c), (f), (i), and (l)). On the other hand, the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' often misclassify a large number of background pixels as foreground ones (Figs. 5.8(h), 5.9(h), 5.12(h), 5.13(h)). Therefore, a large number of reference pixels of each input image corresponding to misclassified background pixels

(a) *Nrgb*_Thresh_Fld290 (b) *Nrgb*_Forgnd_Fld290 (c) *Nrgb*_Backgnd_Fld290

(d) *Nrgb*_Thresh_Bijan1_262 (e) *Nrgb*_Forgrnd_Bijan1_262 (f) *Nrgb*_Backg_Bijan1_262

(g) *Nrgb*_Thrsh_Highway I_261 (h) *Nrgb*_Forg_Highway I_261 (i) *Nrgb*_Back_Highway I_261

(j) *Nrgb*_Thresh_Lab220 (k) *Nrgb*_Forgnd_Lab220 (l) *Nrgb*_Backgnd_Lab220

Fig. 6.7 – Thresholded, foreground, and background frames due to applying the 'Object-based Selective Update using Temporal Median' to Fld290 ((a)-(c)), Bijan1_262 ((d)-(f)), Highway I_261((g)-(i)), and Lab220 ((j)-(l)) all based on normalised *rgb* (*Nrgb*).

are not considered in the 'Temporal Averaging' or the 'Temporal Median' updating process. Rather they are replaced with corresponding pixels from previous background frame. As a result, for a large number of reference pixels of each input frame in the latter methods, computed background pixels are out of date due to copying from previous frame. However, remaining background pixels are considered in the updating process and are up to date. In total, the 'Object-based Selective Update using Temporal Median' produces background frames, which are more precise and have better qualities than produced by the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median'. The quality figures in Table 6.7 confirm this result.

- Based on the criteria of comparison:

  - Good performance:

    Both the 'Pixel-based' and the 'Object-based Selective Update' methods can operate in outdoor and indoor scenes continuously. The sample frames of each method were already shown in Figs. 5.8 to 5.15 and Fig. 6.7. The DRs of the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' are as high as detection rates of the 'Object-based Selective Update using Temporal Median'. But the former methods have higher FARs than the latter method. Thus, the latter method has a better performance than the former ones.

  - Correctness:

    The 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-

The Name of Selective Update Method	***Fld***	***Bijan1***	***Highway I***	***Lab***
The Pixel-based Selective Update using Temporal Averaging	**82**	**73**	**85**	**80**
The Pixel-based Selective Update using Temporal Median	**81**	**69**	**89**	**78**
The Object-based Selective Update using Temporal Median	**97**	**96**	**96**	**93**

Table 6.7 – The quality figures from Table 5.5 for the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median'. The quality figures assigned based on the visual qualities of foreground frames to background images in Fig. 6.7 for the 'Object-based Selective Update using Temporal Median'. All results are based on normalised *rgb*.

based Selective Update using Temporal Median' use separate timers for the pixels of a background image. If a moving object becomes motionless, pixel timers of the object may not have the same value. Thus, not all the parts of the object may become stationary in one frame. Meanwhile, when a stationary object starts moving, the above methods remove the ghost after LT frames in a number of frames (but not in one frame). However, the 'Object-based Selective Update using Temporal Median' utilises an advanced moving status method (Fig. 6.3 and Table 6.3) for identifying the status of moving or stationary objects and ghosts. When an object becomes motionless, it is not shown in the background frame unless *Object-Stationary-Limit* numbers of frames are elapsed. Then, the motionless object is shown in a complete form in one frame (i.e. not gradually in a number of background frames). Meanwhile, the latter method removes ghosts in one background frame.

Therefore, based on the above discussion, the 'Object-based Selective Update using Temporal Median' produces correct background frames. However, the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' methods do not produce correct background frames when a moving object becomes motionless or a stationary object starts moving. Because in both cases either parts of an object are shown or the pixels of a ghost are removed in a number of background frames.

- The speed of computations:

  By comparing the computation times in Tables 5.4 and 6.4 for four video sequences *Fld*, *Bijan1*, *Highway I*, and *Lab*, it is obvious that the 'Object-based Selective Update using Temporal Median' is much faster than the 'Pixel-based Selective Update using Temporal Averaging' and the 'Pixel-based Selective Update using Temporal Median' methods as well.

6.5.3 Comparison of the 'Object-based' approach with related algorithms

The selected algorithms for comparisons are Horprasert et al. (1999) and Kim et al. (2005) which are shortly described in Appendix A.

6.5.3.1 Comparison of the 'Object-based' approach with the Horprasert's algorithm

The Horpraset's algorithm can be compared with the 'Selective Update Using Temporal Median' method as follows:

- Based on quantitative comparison:

  There is no specific approach in the Horprasert's algorithm for computing the

parameter τ_{alo} in Eq. A.7. Thus, this parameter should be given to the algorithm as an input value. Based on Eq. A.7, τ_{alo} has a major role in classifying foreground, background, shadow, and highlight pixels. However, it is not determined what value is suitable to be entered for τ_{alo}. After a number of experiments, it was found that τ_{alo} should be a negative number, for example, between -1.0 to -15.0. Unfortunately, by changing τ_{alo} for a video sequence, different results are obtained. In this regard, their corresponding sample images are given in qualitative comparison part of this section. Meanwhile, a suitable range of values for τ_{alo} varies for each video sequence. Therefore, it is impossible to find specific DR and FAR values for each video sequence as by varying τ_{alo}, the numbers of foreground, background, shadow, and highlight pixels change considerably. Nevertheless, those values of τ_{alo} for eight video sequences are chosen where the best classifications of the pixels of input images are obtained. For such τ_{alo} values, the results of DRs and FARs of the Horperasert's algorithm are given in Table 6.8. Meanwhile, the same sets of frames of eight video sequences, already used for the 'Object-based Selective Update using Temporal Median', were utilised for the Horperasert's algorithm. Based on the results of Table 6.8, it is clearly obvious that DRs and FARs of the Horperasert's algorithm are respectively almost the same and much

Image Sequence	***The Horpraserts's Algorithm***				
	TP	**FP**	**FN**	**DR%**	**FAR%**
Fld	945829	171478	46554	95.31	15.35
Bijan1	1278320	405964	79305	94.12	24.10
Highway I	1388084	192513	309963	81.75	12.18
Highway II	301279	17190	20328	94.60	5.40
Cp1	209806	7898676	0	100.00	97.41
Cp3	193029	7897676	1618	99.17	97.61
Cp5	229108	7857310	2427	98.85	97.17
Lab	636555	9676764	0	100.0	93.83

Table 6.8 - The DR and FAR values of the Horperasert's algorithm applied to eight video sequences.

higher than their corresponding DRs and FARs of the 'Object-based Selective Update using Temporal Median' method. In this case, definitely the 'Object-based Selective Update using Temporal Median' produces much more precise background frames than the Horperasert's algorithm.

- Based on qualitative comparison:

  For qualitative comparison, first it is shown that by varying τ_{alo}, different results are obtained. Sample frames corresponding to three τ_{alo} values for *Fld* and *Bijan1* video sequences are shown in Fig. 6.8 ((a) to (g)). In addition, sample frames for six other video sequences are shown in Fig. 6.9. These sample frames are obtained based on those τ_{alo} values that produce the best classifications of the pixels of each input frame. Their corresponding background frames for eight video sequences due to the 'Object-based Selective Update using Temporal Median' were already shown in Figs. 6.4 to 6.6.

(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

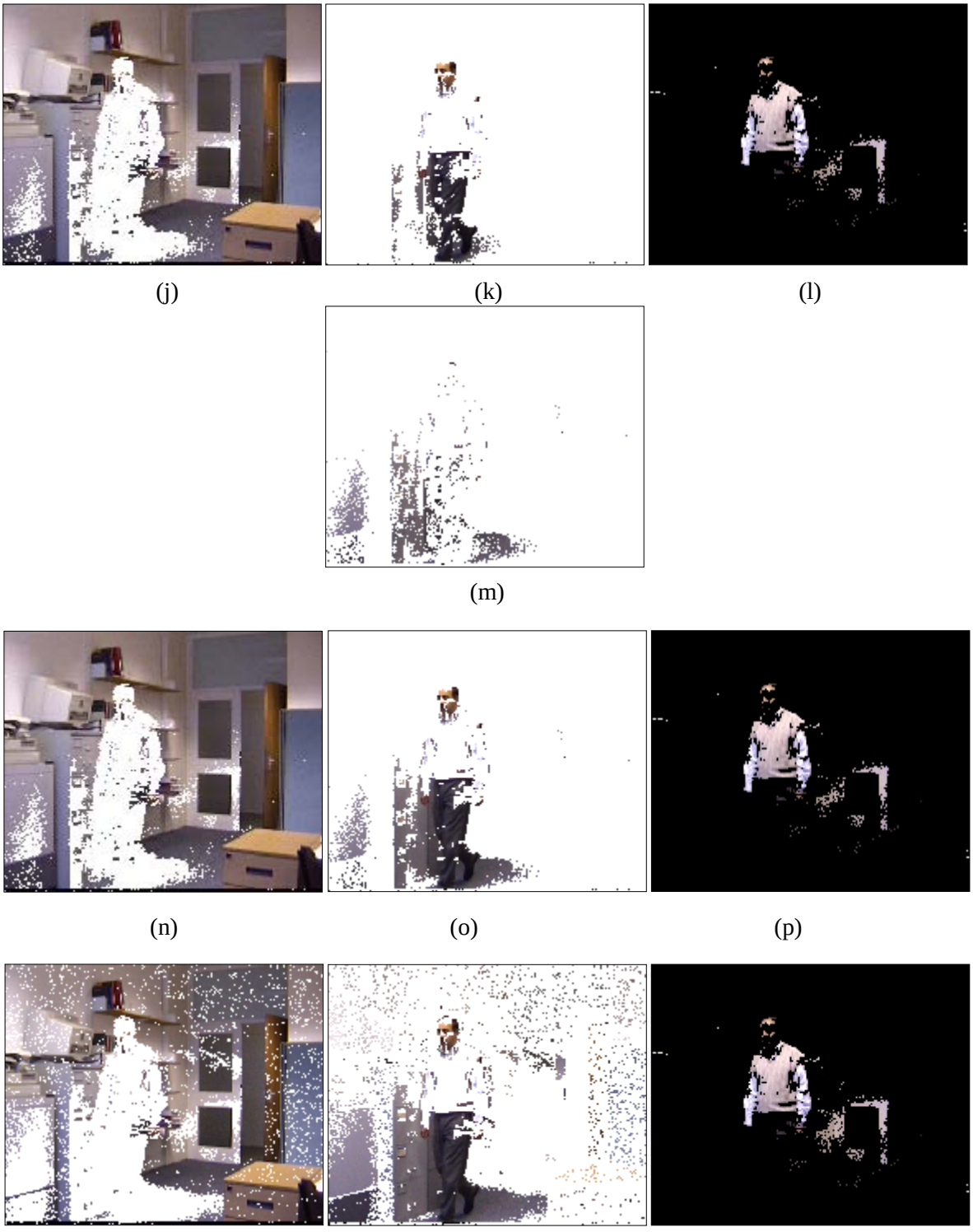


Fig. 6.8 – Background, foreground, and highlight frames produced by applying the Horprasert's algorithm to Fld290 for τ_{alo} = -9.5 ((a)-(c)), for τ_{alo} = -4.5 ((d)-(f)), for τ_{alo} = -1.5 ((g)-(i)) and to Bijan1_262 for τ_{alo} = -10 ((j)-(l), (m) shadow frame), for τ_{alo} = -5 ((n)-(p)), and for τ_{alo} = -2 ((q)-(s)).

(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

(m) (n) (o)

(p) (q) (r)

Fig. 6.9 – Background, foreground, and highlight frames produced by applying the Horprasert's algorithm to Highway I_261 for $\tau_{\alpha lo}$ = -1 ((a)-(c)), to Lab220 for $\tau_{\alpha lo}$ = -10 ((d)-(f)), to Highway II_126 for $\tau_{\alpha lo}$ = -1.5 ((g)-(i)) and to Cp1_381 for $\tau_{\alpha lo}$ = -10 ((j)-(l)), to Cp3_149 for $\tau_{\alpha lo}$ = -5 ((m)-(o)), and to Cp5_139 for $\tau_{\alpha lo}$ = -15 ((p)-(r)).

Total foreground, shadow, and highlight pixels of each input frames in the Horperasert's algorithm correspond to foreground pixels (or regions) of each input frame of the 'Object-based Selective Update using Temporal Median' method. Based on Figs. 6.8 and 6.9, it is obvious that the background frames produced the Horprasert's algorithm have very low qualities even for the best $\tau_{\alpha lo}$ values. On the other hand, Figs. 6.4 to 6.6 show the superior qulity of the background frames produced the 'Selective Update Using Temporal Median' method.

- Based on the criteria of comparison:

  - Good performance:

    In an initial background training process, the Horprasert's algorithm computes some parameters associated with normalisation (i.e. s_i, a_i and b_i) and thresholding (i.e. τ_{CD}, $\tau_{\alpha 1}$ and $\tau_{\alpha 2}$) over a number of static background frames automatically. Except one parameter defined as $\tau_{\alpha lo}$, Horprasert's algorithm has this good performance that the computations of its parameters are automatic. However, this approach has the following problem:

Horprasert et al. (1999) claim that their algorithm is robust and reliable in outdoor scenes. Suppose that in the initial background training process, normalisation (i.e. s_i, a_i and b_i) and thresholding parameters (i.e. τ_{CD}, $\tau_{\alpha 1}$ and $\tau_{\alpha 2}$) are computed in the morning. Thus, there is no guarantee that in an outdoor scene, these thresholds are still valid at noon, in the afternoon or at night. As a result, many pixels may be misclassified which leads to low DRs and high FARs in outdoor environments. Our experiments show the algorithm's DR decreases as the time passes from the time that the thresholds were computed. Figs. 6.8 and 6.9 illustrate the low performance of Horprasert's algorithm for both indoor and outdoor environments. However, the high performance of the 'Selective Update Using Temporal Median' was already shown for eight video sequences in Figs. 6.4 (h and o), 6.5 (g, n and u) and 6.6 (g, n and u).

- Correctness:

  The Horprasert's algorithm can be compared with the 'Selective Update Using Temporal Median' from the correctness point of view as follows:

  Suppose that in the succeeding frames after the training process, a background object starts moving. The reference frame and the thresholds are already computed and are fixed. What happens is that some of the pixels in the initial position of the object are now replaced with new pixels in the input image. Those pixels are different from their corresponding pixels in the reference frame. Thus, the algorithm does not classify them as background pixels. As the object continues its motion, more such pixels are misclassified and neither the reference frame nor the thresholds are updated.

  A similar difficulty occurs when a moving object becomes stationary. It is not included in the background image and the thresholds are not updated as well.

  Therefore, Horprasert's algorithm has major correctness problems. However, the 'Selective Update Using Temporal Median' is able to remove ghosts and include stationary objects in the background image effectively (see Fig. 6.10).

- The speed of computations:

  From the speed of computations' point of view, Horprasert's algorithm has the following advantage:

  It runs very fast and is completely real-time. However, the 'Selective Update Using Temporal Median' is not as fast as Horprasert's algorithm but still runs close to real-time.

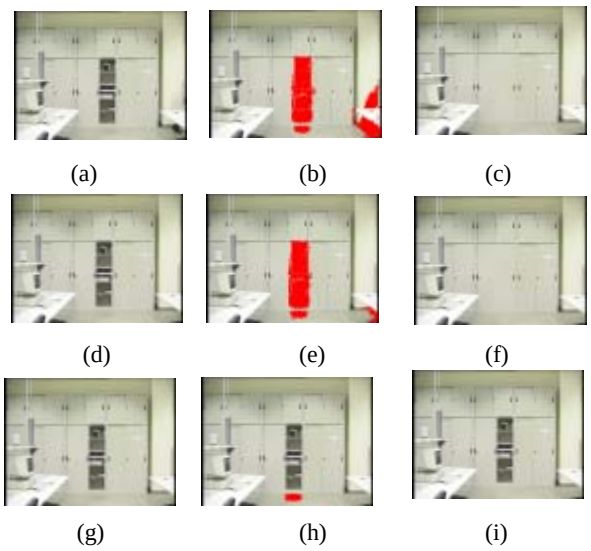


Fig. 6.10 – (a), (d) and (g) Input images Lab376, Lab379, Lab380; (b), (e) and (h) Lab_Fgnd376, Lab_Fgnd379, Lab_Fgnd380; (c), (f) and (i) Corresponding Lab_back376, Lab_back379, Lab_back380, respectively. The ghost in whole is replaced after the fifth background frame from Lab_back376 (i.e. Lab_back380) shown in (i) (one of the novelties of the 'Object-based' approach given in Table 6.3).

6.5.3.2 Comparison of the 'Object-based' approach with the Kim's algorithm

The Kim's algorithm (2005) can also be compared with the 'Selective Update Using Temporal Median' method as follows:

- Based on quantitative comparison:

  The Kim's algorithm encodes the background on a pixel-by-pixel basis. For each pixel, it builds a codebook consisting of one or more codewords. In the training period, input pixels are clustered into the set of codewords based on a colour distortion metric together with brightness bounds. In oreder to construct coodbooks for pixels in the training period and to perform background subtraction in the continuous operation, the Kim's algorithm uses for parameters ε_1, ε_2, α, β. These parameters have essential roles in classifying pixels and should be tuned for each video sequence manually.

  For quantitative comparison, the Kim's algorithm is applied to eight video sequences based on the best tuning of its parameters and its DR and FAR results are given in Table 6.9. Based on the results of Table 6.9, it is clearly obvious that DRs and FARs of the Kim's algorithm are respectively lower and much higher than their corresponding DRs and FARs of the 'Object-based Selective Update using Temporal Median' method. Thus, the 'Object-based Selective Update using Temporal Median' produces much more precise background frames than the Kim's algorithm.

Image Sequence	*The Kim's Algorithm*				
	TP	**FP**	**FN**	**DR%**	**FAR%**
Fld	738285	114561	88702	89.27	13.43
Bijan1	458004	224473	558389	45.06	32.89
Highway I	1698060	166705	476851	78.07	8.94
Highway II	97037	45344	30328	76.19	31.85
Cp1	5948	2234	13091	31.24	27.30
Cp3	1128	586	12011	8.59	34.19
Cp5	29253	8236	14205	67.31	21.97
Lab	135936	17837	447036	23.31	11.60

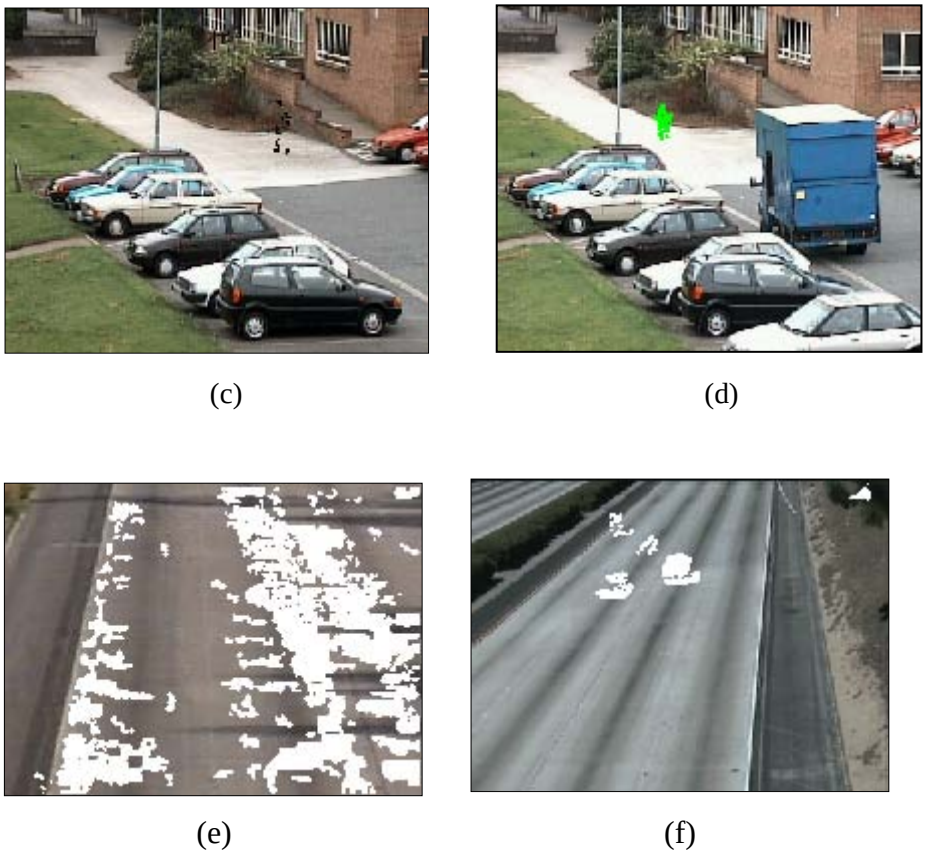
Table 6.9 - The DR and FAR values of the Kim's algorithm applied to eight video sequences.

- Based on qualitative comparison:

For qualitative comparison, the Kim's is applied to eight video sequences. Their corresponding Sample frames are shown in Fig. 6.11 ((a) to (g)). These sample frames are obtained best tuning of four parameters parameters ε_1, ε_2, α, β. Define

(a)

(b)



(c) (d)

(e) (f)

Fig. 6.11 – The Kim's algorithm foreground regions are shown on background frames.

Based on Figs. 6.11, it is obvious that the background frames produced the Kim's algorithm have very low qualities. On the other hand, Figs. 6.4 to 6.6 show the superior qulity of the background frames produced the 'Selective Update Using Temporal Median' method.

In comparison with the Kim's algorithm, the parameters of the 'Selective Update Using Temporal Median' are either fixed or automatically computed for all applications and also has much faster running close to real-time.

Chapter 7

Foreground Boundary Smoothing

7.1 Introduction

The goal of this chapter is to present a simple and effective silhouette (boundary) smoothing method.

7.2 A practical RLE smoothing method

Due to simplicity of processing, binary contours are used to represent and classify patterns of interest in many computer vision applications. However, after thresholding, the boundaries of foreground regions are often corrupted by noisy pixels, which makes direct analysis and feature extraction of these regions too complicated. Obviously smoothing of object boundaries makes a number of their measurements such as perimeter, area, moments, etc easier and more reliable. As stated in Section 2.2.11.1, smoothing is more effective when is performed on binary contours.

A review of smoothing methods for binary contours was offered in Section 2.2.11.1. It is observed that almost all smoothing methods are appropriate for chain code applications. Only RLSA is utilised for RLE representation. However, based on the description of RLSA given in Section 2.2.11.3, it is concluded that RLSA is actually used as a document segmentation tool. It is not utilised as a general purpose method for smoothing the boundaries of foreground regions in binary images. However, the RLE technique has been selected as a more suitable approach for representing foreground binary regions in this thesis. Thus, the lack of an appropriate boundary smoothing method which operates using RLE data will be obvious. For this purpose, a simple RLE smoothing method is proposed in the next subsections.

7.2.1 The general idea of an RLE smoothing method

The idea of a simple smoothing method for RLE data is proposed in this subsection

to represent the foreground regions of a binary image. The goal is to smooth horizontal, vertical and diagonal one-pixel-width noisy pixels along the boundary of foreground regions. Such anomalies called out-spikes in this thesis may appear on the left, on the right, on the top or on the bottom side of a foreground region. In practice, complex combinations of horizontal, vertical and diagonal out-spike may occur on the boundaries of foreground regions. In this case, smoothing is simultaneously performed as the contour of a foreground region is traced. Once an out-spike is found on a segment of the contour, it is smoothed. Then tracing is continued by considering the smoothed segment. This new idea describing how out-spikes will be removed from a region boundary is as follows:

Suppose there is a horizontal one-pixel-width out-spike on the left side of the current row whose starting column begins before the starting columns of its previous and following rows. Thus, the starting column of the current row is set equal to the starting column of one of the mentioned rows with a smaller starting column. Otherwise, in the case of no out-spike, the starting column of the current row remains unchanged.

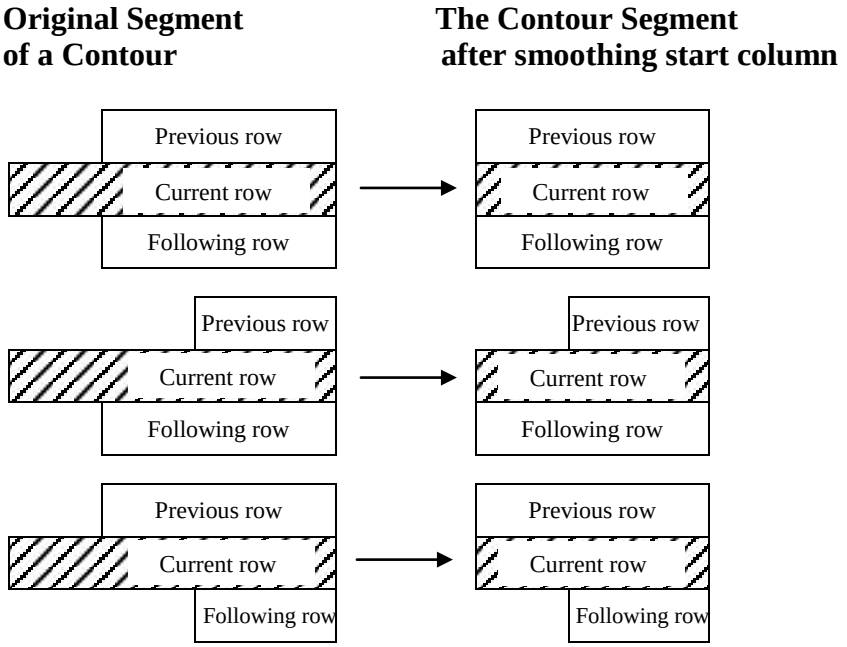
A similar approach is taken for a horizontal one-pixel-width out-spike, which occurs on the right end of the current row. That is, the out-spike end column is set equal to the end column of one of its previous or following rows with a greater end column. Otherwise, in the case of no out-spike, the end column of the current row remains unchanged. Thus, each smoothing of either the left or the right sides of the current row is called "*row smoothing*" (see Fig. 7.1). If several parts of a foreground binary region have distinct out-spikes on a row, row smoothing will apply to all of them, as they are detected by the contour tracing algorithm traversing the region boundary. *Row smoothing* can be stated using the following pseudo code:

```
if (start_col of current_row < start_col of previous_row  and
    start_col of current_row < start_col of following_row)

    if (start_col of previous_row < start_col of following_row)
        start_col of current_row ← start_col of previous_row
    else
        start_col of current_row ← start_col of following_row

if (end_col of current_row > end_col of previous_row  and
    end_col of current_row > end_col of following_row)

    if (end_col of previous_row > end_col of following_row)
        end_col of current_row ← end_col of previous_row
    else
        end_col of current_row ← end_col of following_row
```

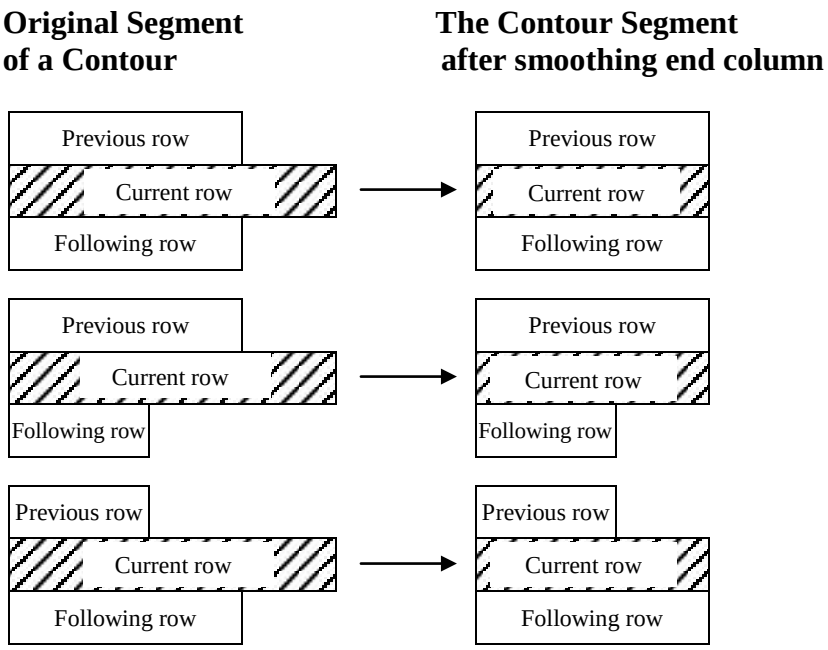
This approach is similarly applied to vertical columns. This stage is called "*column smoothing*". By column smoothing, the runs of a vertical out-spike with the length of one pixel are eliminated from the RLE data structure.

The smoothing of the left and the right sides of the current row is illustrated

in Fig. 7.1a and Fig. 7.1b, respectively. Vertical smoothing of top and bottom sides can be easily observed by rotating Fig. 7.1a and Fig. 7.1b clockwise.



(a)



(b)

Fig. 7.1 – (a) The starting column of out-spike row (current row) is set to the starting column of one of its previous or following rows with smaller starting column after smoothing. (b) The end column of out-spike row (current row) is set to end column of one of its previous and following rows with greater end column after smoothing.

In addition to the row and column smoothing, it is sometimes possible that a combination of one or two diagonal pixels may appear on the corners of some horizontal or vertical segments. Thus such pixels may occur diagonally in the directions of 1, 3, 5, 7 of Freeman codes. This stage, which removes such pixels, is called "*diagonal smoothing*".

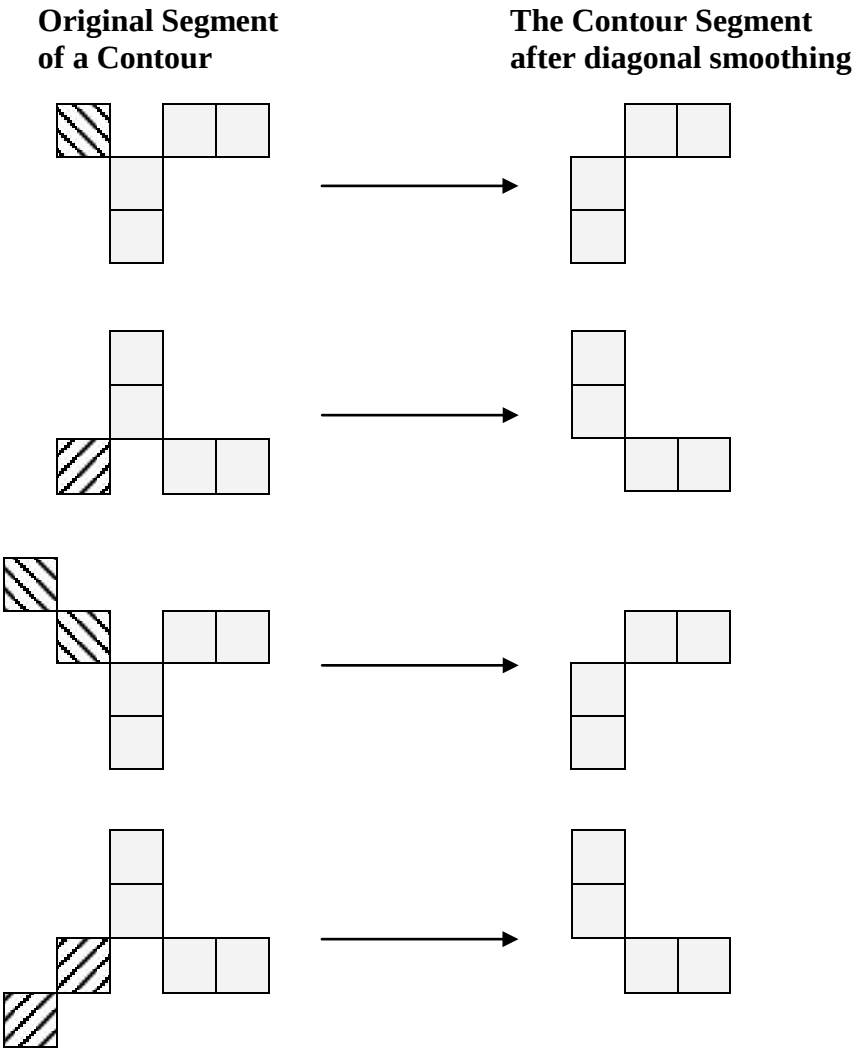


Fig. 7.2 – Typical diagonal smoothing of one and two out-spikes in the directions of 3 and 5.

The general idea of RLE smoothing was explained above. However, in order to implement this idea, some specific codes and structures should be utilised. As a result, all complex types of out-spikes of region boundaries can be effectively removed as stated in the next subsections.

7.2.2 A suitable data structure for boundary smoothing

A smoothing method for Quek's algorithm (2000) will be the proposed RLE smoothing algorithm in this thesis. Obviously the proposed smoothing algorithm should completely match with it. In Quek's algorithm, RLE boundary tracing is achieved by

jumping from one run to another. This is performed using the "WALK$_l$" and "WALK$_r$" functions in which some portions of runs are often traversed.

Fig. 7.3 shows the transition table of Quek's algorithm for the 4-state finite state machine to compute region boundaries. In this figure, due to symmetry, tests and operations of rows 1 and 4 and rows 2 and 3 are the same. Thus, by ignoring rows 3 and 4 only rows 1 and 2 will be considered. Besides, for easily referencing the states in Fig. 7.3, the states in rows 1 and 2 are numbered from left to right by 1 to 8, respectively. Meanwhile, the "WALK$_l$" function of state 8 in Fig. 7.3 and the right

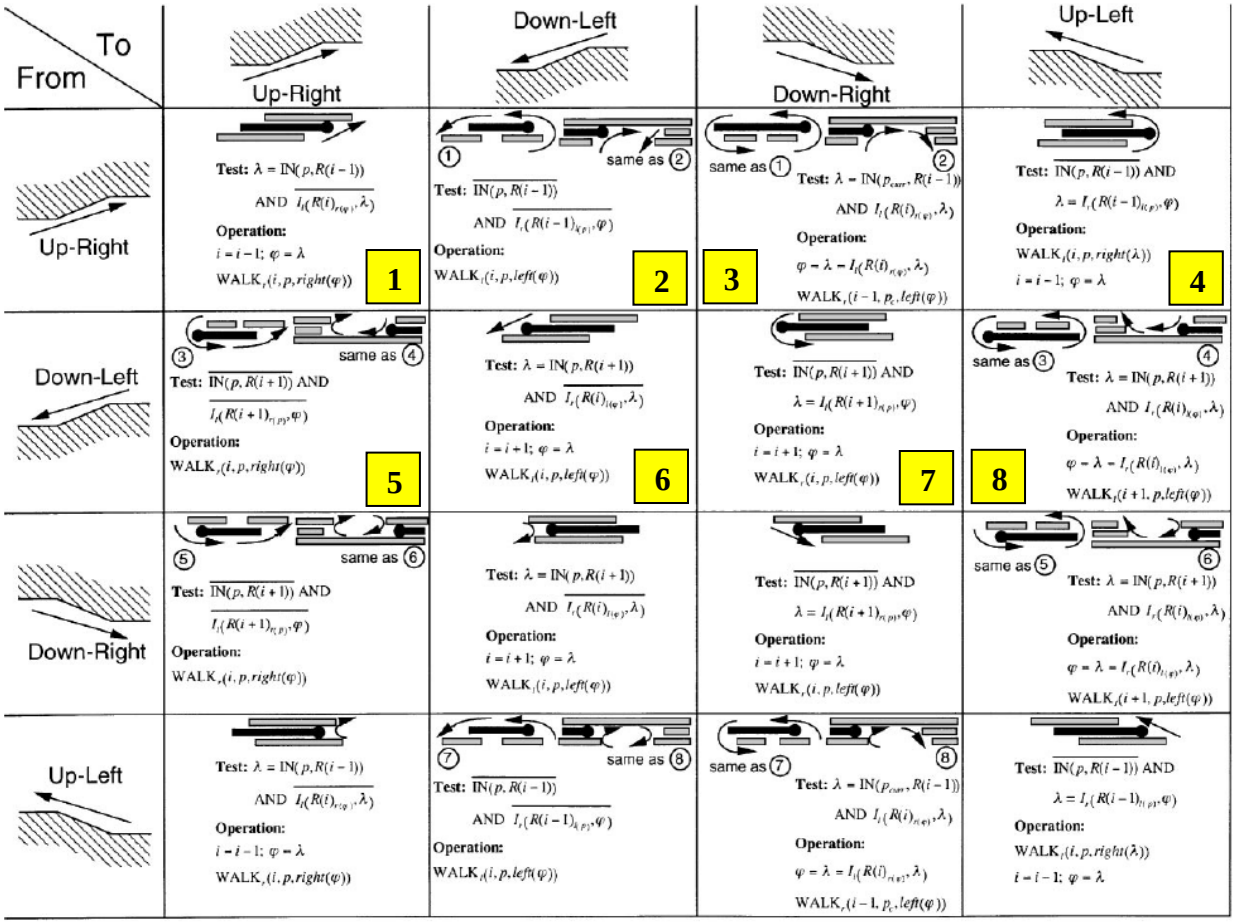


Fig. 7.3 – Transition table for the 4-state finite state machine to compute region boundaries (from Quek (2000)). The states in the first two rows are numbered by 1 to 8 for easier referencing.

Incorrect Predicate/State	Correct Predicate/State
$\beta \equiv I_r(R(i)_{r(\psi)}, \lambda)$	$\beta \equiv I_r(R(i)_{l(\psi)}, \lambda)$
$\gamma \equiv I_l(R(I-1)_{r(p)}, \psi)$	$\gamma \equiv I_l(R(i+1)_{r(p)}, \psi)$
In state 8: WALK$_l$ (i+1, p, *left*(ψ))	In state 8: WALK$_l$ (i+1, p, *right*(ψ))

Table 7.1 – The correct forms of Quek's transition table predicates and state 8.

sides of the equations for two predicates β and γ on page 1643 contain typing mistakes which are given in Table 7.1 in accompany with their corrections. However, the predicate values for β and γ in the "Test" sections of states 5 to 8 in Fig. 7.3 are correct.

Each state from 1 to 8 in Fig. 7.3 has one or more figures showing the run configuration. In these states, the solid black run represents the current run ψ. In addition, the current point *p before* the transition is marked by the black dot '●' on ψ. Meanwhile, each state contains a "Test" and an "Operation" section where the latter consists of a "WALK$_l$" or a "WALK$_r$" function (Quek (2000)). Meanwhile, in Quek's boundary tracing, a transition is often done from run *i* to run *j*. It is also possible that runs *i* and *j* are the same and the transition is performed from run *i*'s left side end to its right end or vice versa.

Although the smoothing rules offered in section 7.2.1 seem quite simple, an appropriate tool is necessary to implement those rules effectively. In this regard, for representing the transitions in "WALK$_l$" and "WALK$_r$" functions and for smoothing a region boundary, a suitable structure is required. For this purpose, a data structure called boundary code (inspired from Hu et al.'s (1998) paper) is proposed in this section. A data structure of a boundary code based on 8-connectivity is similar to a vector with the following parameters:

– st_p_x, // vector start point column (x) value on run *i*
– st_p_y, // vector start point row (y) value on run *i*
– en_p_x, // vector end point column (x) value on run *j*
– en_p_y, // vector end point row (y) value on run *j*
– st_run, // vector start run *i*
– en_run, // vector end run *j*
– direc, // vector Freeman direction from run *i* to run *j*
– m, // current row in the present foreground region
– next_st, // next state in the transition table
– cur_run, // current run
– i_val, // array index of the current vector

Based on the proposed data structure, the boundary traversals performed in "WALK$_l$" and "WALK$_r$" states 1 to 8 in Fig. 7.3 can be expressed using boundary codes in the following paragraphs. Besides, in addition to black dot '●' stated above, it is assumed that the white dot '○' shows the end of transition in the "WALK$_l$" and "WALK$_r$" functions. The column, row, and the run of the black dot '●' are called here as p_col, p_row, p_run, respectively. The column, row, and the run of the white dot '○' are also called as end_point_col, end_point_row, end_point_run, respectively.

In states 1 to 8 of Fig. 7.3, the run configurations and transitions, shown below, occur as follows:

- State 1: **Test:** λ = IN (p, $R(i-1)$) AND $\overline{I_l(R(i)_{r(\varphi)}, \lambda)}$
  Operation: $i = i - 1$; $\varphi = \lambda$; WALK$_r$ (i, p, *right*(φ))

```
if (end_point_col = p_col)  // Fig. 7.4(a)
{
   st_p_x ← p_col; st_p_y ← p_row; en_p_x ← p_col; en_p_y ← end_point_row;
   st_run ← p_run; en_run ← end_point_run; direc ← 2;
}
else if (end_point_col = p_col + 1)  // Fig. 7.4(b)
{
    st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← p_col + 1;
    en_p_y ← end_point_row;  st_run ← p_run;  en_run ← end_point_run;
    direc ← 1;
}
else if (p_row = end_point_row)  // Fig. 7.4(c)
{
    st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col;
    en_p_y ← end_point_row;  st_run ← p_run;  en_run ← p_run;  direc ← 0;
}
else
{
    // Transition 1 shown in Fig. 7.4(d).
    st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← p_col + 1;
    en_p_y ← end_point_row; st_run ← p_run; en_run ← end_point_run;
    direc ← 1;

    // Transition 2 shown in Fig. 7.4(d).
    st_p_x ← p_col + 1;  st_p_y ← end_point_row;  en_p_x ← end_point_col;
    en_p_y ← end_point_row; st_run ← end_point_run; en_run ← end_point_run;
    direc ← 0;
}
```

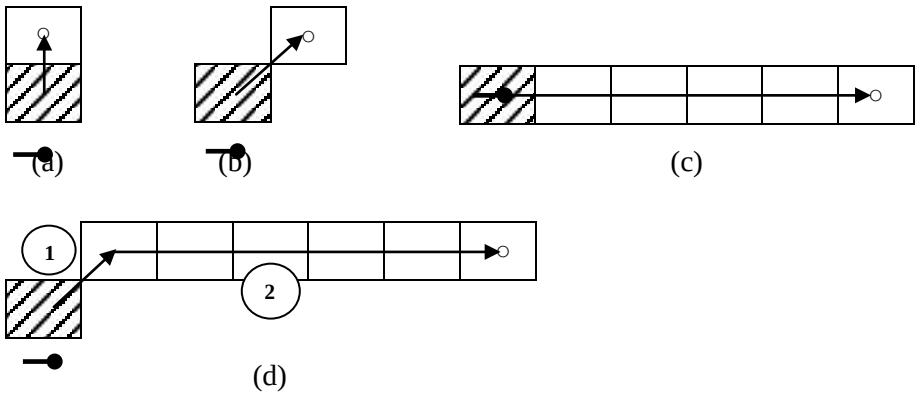


Fig. 7.4 – (a) end_point_col = p_col (b) end_point_col = p_col + 1
(c) p_row = end_point_row (d) none of (a) to (c) conditions

- State 2: **Test:** $\overline{IN(p,\ R(i-1))}$ AND $\overline{I_r(R(i-1)_{l(p)}, \varphi)}$
  Operation: WALK$_l$ (i, p, *left* (φ))

```
if (p_row = end_point_row)  // Fig. 7.5(a)
{
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row; st_run ← en_run ← p_run;  direc ← 4;
}
else if (p_col = end_point_col + 1)  // Fig. 7.5(b)
{
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row;  st_run ← p_run;  en_run ← end_point_run;
   direc ← 5;
}
else
{
   // Transition 1 shown in Fig. 7.5(c).
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col + 1;
   en_p_y ← p_row;  st_run ← p_run; en_run ← p_run;  direc ← 4;

   // Transition 2 shown in Fig. 7.5(c).
   st_p_x ← end_point_col + 1;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row;  st_run ← p_run;  en_run ← end_point_run;
   direc ← 5;
}
```

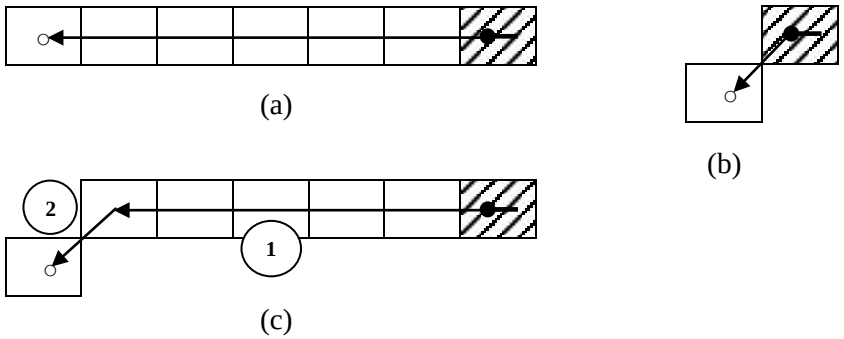


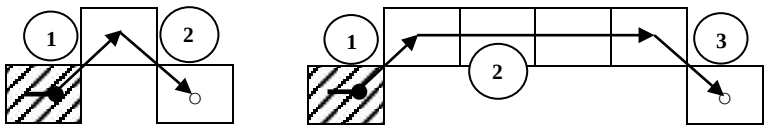
Fig. 7.5 – (a) p_row = end_point_row (b) p_col = end_point_col + 1
(c) none of (a) and (b) conditions

- State 3: **Test:** λ = IN (p, $R(i-1)$) AND $I_l(R(i)_{r(\varphi)}, \lambda)$

  Operation: $\varphi = \lambda = I_l(R(i)_{r(\varphi)}, \lambda)$; WALK$_r$ ($i-1$, p, *left*(φ))

```
if (end_point_col = p_col + 2)  // Fig. 7.6(a)
{
   // Transition 1 shown in Fig. 7.6(a).
   st_p_x ← p_col; st_p_y ← p_row; en_p_x ← p_col + 1; en_p_y ← p_row – 1;
   st_run ← p_run;  en_run ← mid_run;  direc ← 1;

   // Transition 2 shown in Fig. 7.6(a).
   st_p_x ← end_point_col – 1;  st_p_y ← p_row – 1;  en_p_x ← end_point_col;
```

```
    en_p_y ← end_point_row;  st_run ← mid_run;  en_run ← end_point_run;
    direc ← 7;
}
else
{
    // Transition 1 shown in Fig. 7.6(b).
    st_p_x ← p_col; st_p_y ← p_row; en_p_x ← p_col + 1; en_p_y ← p_row – 1;
    st_run ← p_run;  en_run ← mid_run;  direc ← 1;

    // Transition 2 shown in Fig. 7.6(b).
    st_p_x ← p_col + 1;  st_p_y ← p_row – 1;  en_p_x ← end_point_col – 1;
    en_p_y ← p_row – 1;  st_run ← mid_run;  en_run ← mid_run;   direc ← 0;

    // Transition 3 shown in Fig. 7.6(b).
    st_p_x ← end_point_col – 1;  st_p_y ← p_row – 1;  en_p_x ← end_point_col;
    en_p_y ← end_point_row;  st_run ← mid_run;  en_run ← end_point_run;
    direc ← 7;
}
```



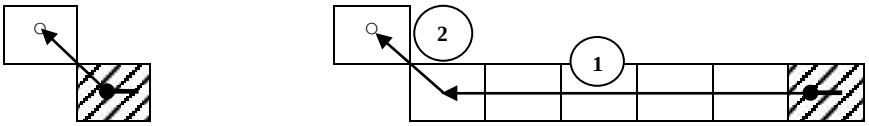
(a) (b)

Fig. 7.6 – (a) end_point_col = p_col + 2 (b) not (a) condition

- State 4: **Test:** $\overline{IN(p,\ R(i-1))}$ AND $\lambda = I_r(R(i-1)_{l(p)}, \boldsymbol{\varphi})$

  Operation: WALK$_l$ (i, p, *right* (φ)); $i = i - 1$; $\varphi = \lambda$;

```
if (end_point_col = p_col – 1)  // Fig. 7.7(a)
{
    st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col;
    en_p_y ← end_point_row; st_run ← p_run; en_run ← end_point_run;
    direc ← 3;
}
else
{
    // Transition 1 shown in Fig. 7.7(b).
    st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col –1;
    en_p_y ← p_row;  st_run ← en_run ← p_run;  direc ← 4;

    // Transition 2 shown in Fig. 7.7(b).
    st_p_x ← end_point_col – 1;  st_p_y ← p_row;  en_p_x ← end_point_col;
    en_p_y ← end_point_run;  st_run ← p_run;  en_run ← end_point_run;
    direc ← 3;
}
```



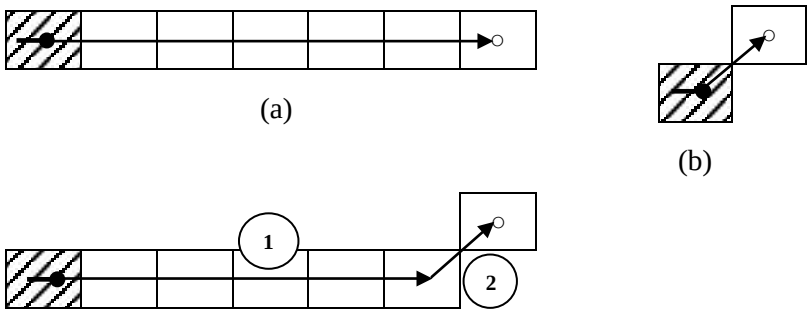
(a) (b)

Fig. 7.7 – (a) end_point_col = p_col – 1 (b) not (a) condition

- State 5: **Test:** $\overline{\text{IN}(p,\ R(i+1))}$ AND $\overline{I_l(R(i+1)_{r(p)},\ \varphi)}$

  Operation: $\text{WALK}_r(i, p, right(\varphi))$

```
if (p_row = end_point_row)  // Fig. 7.8(a)
{
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row; st_run ← p_run;  en_run ← end_point_run;
   direc ← 0;
}
else if (end_point_col = p_col + 1)  // Fig. 7.8(b)
{
   st_p_x ← p_col ;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row;  st_run ← p_run;  en_run ← end_point_run;
   direc ← 1;
}
else
{
   // Transition 1 shown in Fig. 7.8(c).
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col – 1;
   en_p_y ← p_row;  st_run ← en_run ← p_run;  direc ← 0;

   // Transition 2 shown in Fig. 7.8(c).
   st_p_x ← end_point_col – 1;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row;  st_run ← p_run;  en_run ← end_point_run;
   direc ← 1;
}
```



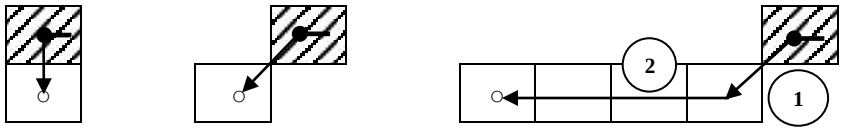
(a) (b) (c)

Fig. 7.8 – (a) p_row = end_point_row (b) p_col = end_point_col + 1
(c) none of (a) and (b) conditions

➢ State 6: **Test:** λ = IN (p, $R(i+1)$) AND $\overline{I_r(R(i)_{l(\varphi)}, \lambda)}$

Operation: $i = i + 1$; $\varphi = \lambda$; WALK$_l$ (i, p, *left*(φ))

```
if (end_point_col = p_col)  // Fig. 7.9(a)
{
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row; st_run ← p_run;  en_run ← end_point_run;
   direc ← 6;
}
else if (end_point_col = p_col – 1)  // Fig. 7.9(b)
{
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row; st_run ← p_run;  en_run ← end_point_run;
   direc ← 5;
}
else
{
   // Transition 1 shown in Fig. 7.9(c).
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col + 1;
   en_p_y ← p_row; st_run ← p_run;  en_run ← p_run;  direc ← 4;

   // Transition 2 shown in Fig. 7.9(c).
   st_p_x ← end_point_col + 1;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row;  st_run ← p_run;  en_run ← end_point_run;
   direc ← 5;
}
```



(a) (b) (c)

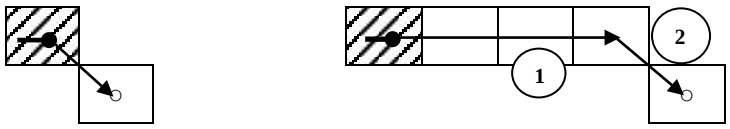
Fig. 7.9 – (a) p_col = end_point_col (b) end_point_col = p_col – 1
(c) none of (a) and (b) conditions

➢ State 7: **Test:** $\overline{IN(p,\ R(i+1))}$ AND $\lambda = I_l(R(i+1)_{r(p)}, \varphi)$

Operation: $i = i + 1$; $\varphi = \lambda$; WALK$_r$ (i, p, *left* (φ));

```
if (end_point_col = p_col + 1)  // Fig. 7.10(a)
{
   st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col;
   en_p_y ← end_point_row; st_run ← p_run;  en_run ← end_point_run;
   direc ← 7;
}
else
{
```

```
    // Transition 1 shown in Fig. 7.10(b).
    st_p_x ← p_col;  st_p_y ← p_row;  en_p_x ← end_point_col –1;
    en_p_y ← p_row;  st_run ← en_run ← p_run;  direc ← 0;

    // Transition 2 shown in Fig. 7.10(b).
    st_p_x ← end_point_col – 1;  st_p_y ← p_row;  en_p_x ← end_point_col;
    en_p_y ← end_point_run;  st_run ← p_run;  en_run ← end_point_run;
    direc ← 7;
}
```



(a) (b)

Fig. 7.10 – (a) end_point_col = p_col + 1 (b) not (a) condition

- State 8: **Test:** λ = IN (p, $R(i+1)$) AND $I_r(R(i)_{l(\varphi)}, \lambda)$

  Operation: $\varphi = \lambda = I_r(R(i)_{l(\varphi)}, \lambda)$; WALK$_l$ ($i+1$, p, $right(\varphi)$)

```
if (end_point_col = p_col – 2)  // Fig. 7.11(a)
{
    // Transition 1 shown in Fig. 7.11(a).
    st_p_x ← p_col; st_p_y ← p_row; en_p_x ← p_col – 1; en_p_y ← p_row + 1;
    st_run ← p_run;  en_run ← mid_run;  direc ← 5;

    // Transition 2 shown in Fig. 7.11(a).
    st_p_x ← end_point_col + 1;  st_p_y ← p_row + 1;  en_p_x ← end_point_col;
    en_p_y ← end_point_row;  st_run ← mid_run;  en_run ← end_point_run;
    direc ← 3;
}
else
{
    // Transition 1 shown in Fig. 7.11(b).
    st_p_x ← p_col; st_p_y ← p_row; en_p_x ← p_col – 1; en_p_y ← p_row + 1;
    st_run ← p_run;  en_run ← mid_run;  direc ← 5;

    // Transition 2 shown in Fig. 7.11(b).
    st_p_x ← p_col – 1;  st_p_y ← p_row + 1;  en_p_x ← end_point_col + 1;
    en_p_y ← p_row + 1;  st_run ← en_run ← mid_run;  direc ← 4;

    // Transition 3 shown in Fig. 7.11(b).
    st_p_x ← end_point_col + 1;  st_p_y ← p_row + 1;  en_p_x ← end_point_col;
    en_p_y ← end_point_row;  st_run ← mid_run;  en_run ← end_point_run;
    direc ← 3;
}
```

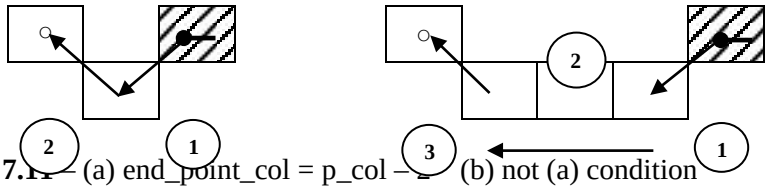


Fig. 7.11 – (a) end_point_col = p_col – 2 (b) not (a) condition

7.2.3 RLE boundary smoothing forms

The details of RLE boundary traversals performed in the "$WALK_l$" and "$WALK_r$" functions with the use of the data structure of a boundary code were offered in section 7.2.2. Once Quek's algorithm traverses the boundary of a foreground region, an array of boundary codes is created. Meanwhile, for each array index value, the other parameters of the data structure of a boundary code including "*m*", "*next-st*", "*cur-run*" and "*i-val*" are determined.

During contour tracing of a foreground region, a number of out-spikes may occur around a region boundary. As stated in section 7.2.1, out-spikes usually have horizontal, vertical or diagonal shapes. Besides, they often have special forms. Fortunately, each of these out-spikes can be described by a specific code number which is determined by a sequence of directions of boundary codes.

Sometimes a horizontal out-spike with a specific form occurs on a segment of a foreground region in a binary image. For example, this might happen when a region boundary is traced from top to bottom. Suppose the boundary of this region is also traced from bottom to top. In this case, it is also possible that an up-side down form of that horizontal out-spike may occur on another segment of this region boundary. This latter form is obtained by rotating the previous horizontal form in counter clockwise direction by 180 degrees. These forms are hereafter called "normal" and "180-rotated horizontal" forms, respectively, for the simplicity of referencing them. Thus, it is possible that "normal" or "180-rotated" forms of horizontal out-spikes may occur on different segments of a region boundary. Besides, in similar situations, "normal" or "180-rotated" forms of vertical or diagonal out-spikes may also appear on the boundaries of foreground regions. Therefore, a number of "normal" and "180-rotated" forms are proposed for recognising horizontal, vertical or diagonal out-spikes. In addition to these out-spike forms, their RLE smoothing forms, based on the ideas given in section 7.2.1, are also offered in the next sections.

7.2.4 Left side and right side RLE boundary smoothing forms

As stated in section 7.2.3, a "normal" horizontal out-spike can be uniquely described by a code number consisting of a sequence of directions of boundary codes. Besides, its "180-rotated" form can also be expressed by another code number. This latter code number is obtained by finding the dual of its Freeman code digits. The dual of a Freeman code is defined as:

$$\text{Dual of a Freeman code} = (\text{Freeman code} + 4) \bmod 8 \qquad \text{(Eq. 7.1)}$$

where a Freeman code = {0, 1, 2, 3, 4, 5, 6, 7} for the 8-connectivity. Thus, if the form of a normal horizontal out-spike is expressed by the code number 617, its 180-rotated horizontal form is stated by 253 (i.e. 617 $\xrightarrow{\textit{dual of each digit}}$ 253).

Although in section 7.2.3 out-spikes were classified into three groups of horizontal, vertical and diagonal, however, in practice out-spikes occur in forms consisting of horizontal, horizontal and diagonal, vertical or vertical and diagonal pixels. Thus, for a more logical classification, it is better to call them hereafter as left side, right side, top side and bottom side out-spikes.

In this section, a number of forms for recognising left side (normal horizontal) out-spikes are proposed which will be specified by their code numbers. In each form (or shape), the out-spike's pixels, which should be removed, are depicted as cross-hatched pixels. The left side forms are illustrated in the left side of a table. In addition, its right side (i.e. "180-rotated") form is also illustrated on the right side of the table. The out-spike's pixels in the latter form are also depicted as cross-hatched pixels.

The smoothed RLE form for each shape (i.e. left side and right side forms) is obtained by removing the cross-hatched pixels. That is, the smoothed RLE boundary consists of only simple grey pixels. The arrows depicted on each shape represent the directions in which Quek's algorithm traces a region boundary. In fact, each out-spike is uniquely specified by the directions of these boundary codes. The index '*i*' on each shape indicates the current position (or index) in the array of boundary codes. Thus bc[i].direc shows the rightmost digit in each out-spike's code number, and bc[i-1].direc, bc[i-2].direc, etc also indicate second rightmost, third rightmost and the other digits in a code number, respectively.

The RLE smoothing method works as follows:

After the process of connected components labelling, the information of foreground binary regions is stored in an RLE data structure. Small noisy regions with areas less than a T_0 threshold are removed using a size filter. These noisy regions are eliminated from the binary image and the RLE data structure as well. Then Quek's RLE boundary tracing algorithm is performed.

As stated in section 7.2.2, the "WALK$_l$" and "WALK$_r$" functions may consist of more than one transition. Thus, for each transition, the array index is increased and the transition's data structure of the boundary code is filled. After the "WALK$_l$" and "WALK$_r$" functions finish and when their corresponding data structures of the boundary codes (at least one) are filled, the RLE smoothing function is called. It will be an incorrect action to call the RLE smoothing method after each transition in the "WALK$_l$" and "WALK$_r$" functions. If this happens, Quek's algorithm will fail to trace the boundary of a region correctly. Therefore, for detecting an out-spike, usually a longer sequence of directions of the boundary codes must be considered. So the direction codes, which specify an out-spike, may appear among the other direction codes. As a result, special code numbers should be used for recognising out-spikes.

The proposed left side and right side (i.e. “normal” and “180-rotated”) forms are illustrated in the following tables. In addition, for a few of them, smoothing pseudo codes show how out-spike’s pixels are removed from the binary image and the RLE data structure.

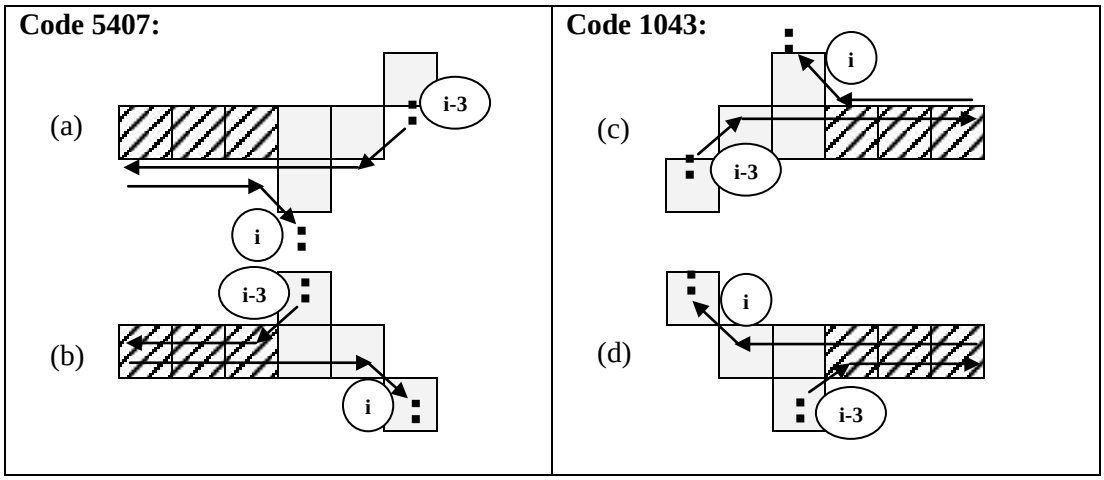


Fig. 7.12 – (a) bc[i].en_p_x < bc[i-3].st_p_x (b) bc[i].en_p_x ≥ bc[i-3].st_p_x
(c) bc[i].en_p_x≥ bc[i-3].st_p_x (d) bc[i].en_p_x < bc[i-3].st_p_x

A pseudo program for the code number 5407 can be written as follows. The reader can easily write a similar pseudo program for the code number 1043 as well.

```
if (bc[i].direc = 7 and bc[i-1].direc = 0 and bc[i-2].direc = 4 and bc[i-3].direc = 5)
{
   if (bc[i-3].st_p_x ≥ bc[i].en_p_x)  // i.e. Code 5407(a)
   {
      y1 ← bc[i-1].st_p_y;

      // remove noisy pixels by making them black on the output image

      for (x1 ← bc[i-1].st_p_x;  x1≤ bc[i-1].en_p_x;  x1 ← x1 + 1)
           output_Image (x1, y1) ← 0;
}
else
{
    // if (bc[i-3].st_p_x < bc[i].en_p_x)  // i.e. Code 5407(b)

     y1 ← bc[i-2].st_p_y;

    for (x1 ← bc[i-2].en_p_x;  x1≤ bc[i-2].st_p_x;  x1 ← x1 + 1)
         output_Image (x1, y1) ← 0;
}
```

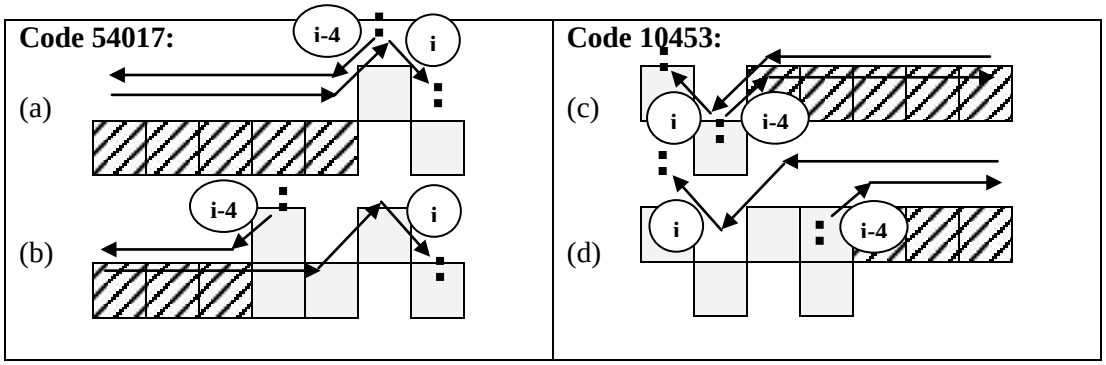


Fig. 7.13 – (a) bc[i].st_p_x = bc[i-4].st_p_x (b) bc[i-4].st_p_x < bc[i].st_p_x
(c) bc[i].st_p_x = bc[i-4].st_p_x (d) bc[i-4].st_p_x > bc[i].st_p_x

// The smoothing pseudo program for the code number 54017

```
if (bc[i].direc = 7 and bc[i-1].direc = 1 and bc[i-2].direc = 0 and bc[i-3].direc = 4
   and bc[i-4].direc = 5)
{
   if (bc[i-4].st_p_x = bc[i].st_p_x)  // i.e. Code 54017(a)
   {
       y1 ← bc[i-2].st_p_y;
      for (x1 ← bc[i-2].st_p_x;   x1≤ bc[i-2].en_p_x;  x1 ← x1 + 1)
           output_Image (x1, y1) ← 0;
}
else
{
    // if (bc[i-4].st_p_x < bc[i].st_p_x)  // i.e. Code 54017(b)

     y1 ← bc[i-3].st_p_y;
      for (x1 ← bc[i-3].en_p_x;   x1≤ bc[i-3].st_p_x;  x1 ← x1 + 1)
           output_Image (x1, y1) ← 0;
}
```

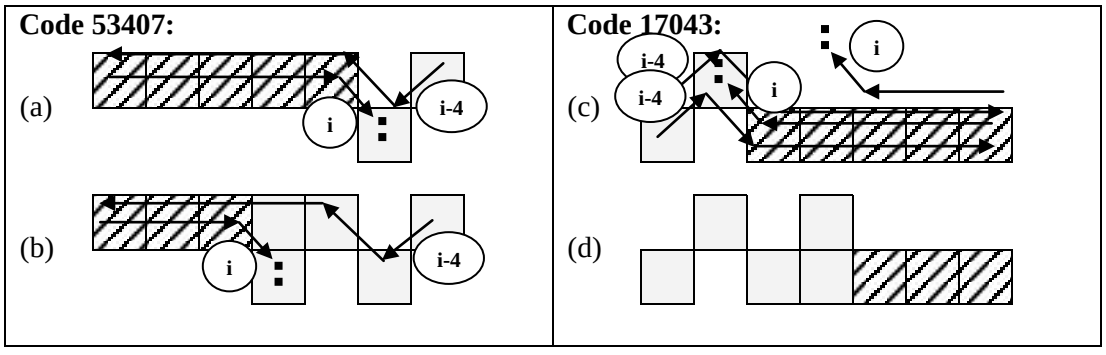


Fig. 7.14 – (a) bc[i].en_p_x = bc[i-4].en_p_x (b) bc[i-4].en_p_x > bc[i].en_p_x
(c) bc[i].en_p_x = bc[i-4].en_p_x (d) bc[i-4].en_p_x < bc[i].en_p_x

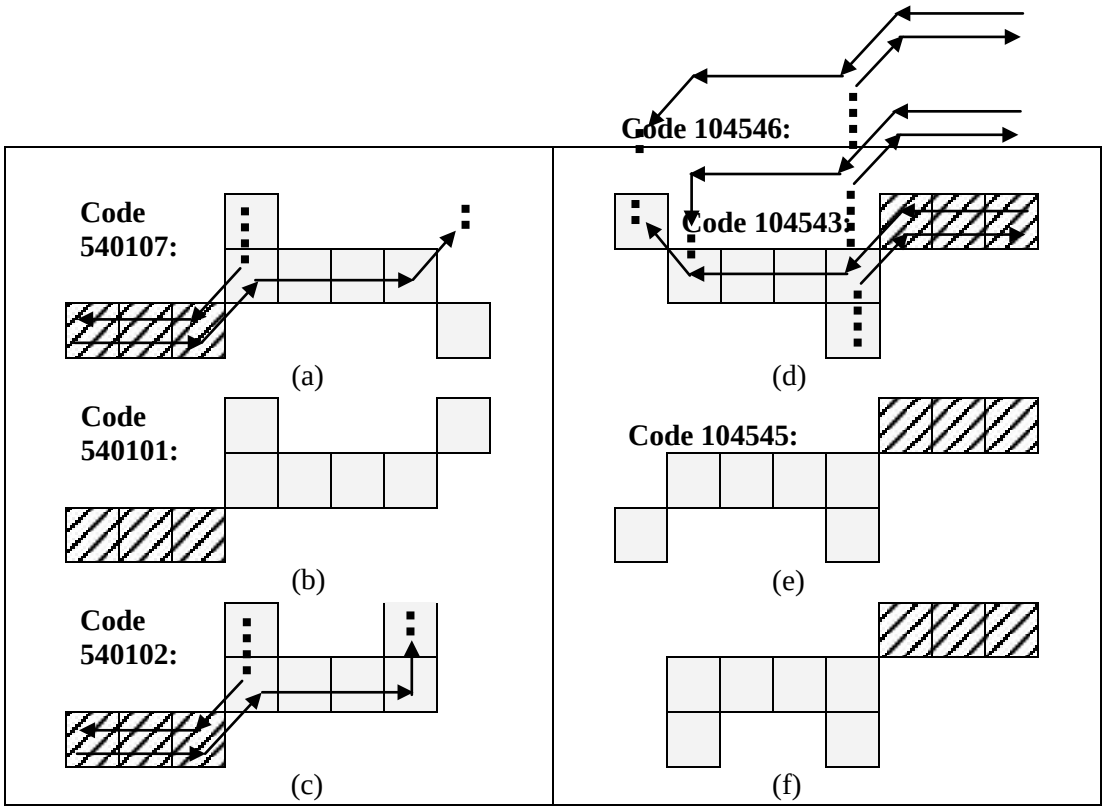


Fig. 7.15 – (a), (b), (c) Code numbers 540107(1)(2), figures (a), (b) and (c) have very similar code numbers so that they only differ in their the rightmost digit shown in the parentheses; (d), (e), (f) Code numbers 104543(5)(6).

Some out-spikes have very similar code numbers so that they only differ in their rightmost or leftmost digit. Fig. 7.15 shows an example for this case. In order to reduce the amount of the required space, the out-spike forms of similar code numbers are combined into one out-spike form hereafter. In this case, for one of the code numbers, the arrow corresponding to its rightmost or leftmost digit is shown in full-line form. For other code numbers, the arrows corresponding to their rightmost or leftmost digits are shown using dashed lines. As an example, consider three out-spike forms in Fig. 7.15, which are combined into one out-spike form, as shown in Fig. 7.16.

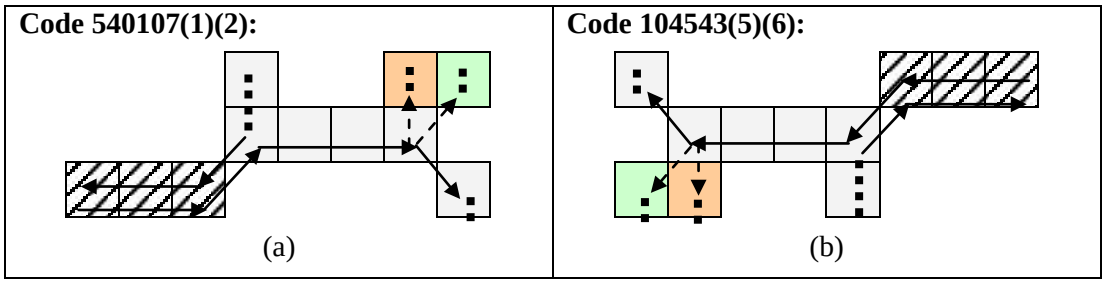
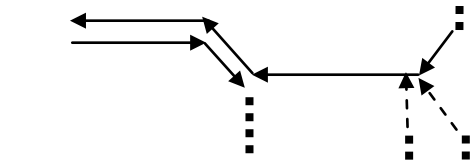


Fig. 7.16 – (a) Code number 540107(1)(2) (b) Code number 104543(5)(6)



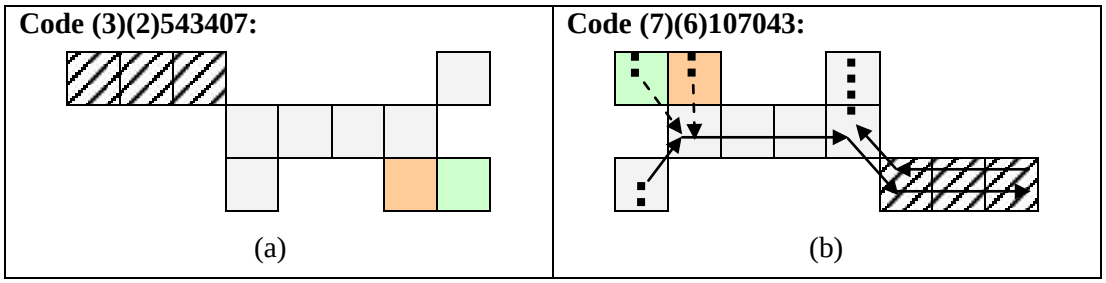


Fig. 7.17 – (a) Code number (3)(2)543407 (b) Code number (7)(6)107043

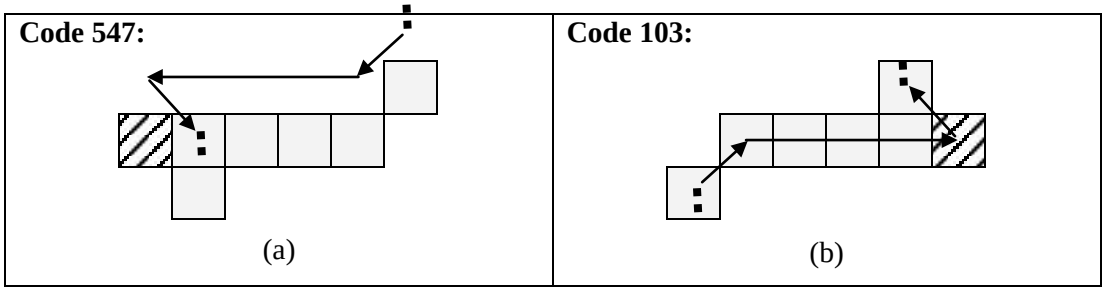


Fig. 7.18 – (a) Code number 547 (b) Code number 103

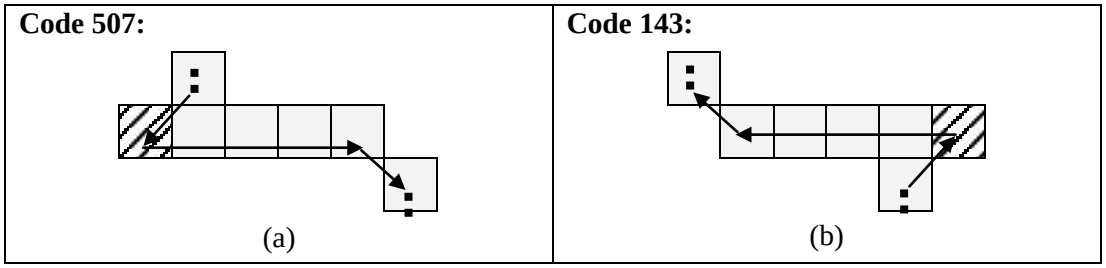


Fig. 7.19 – (a) Code number 507 (b) Code number 143

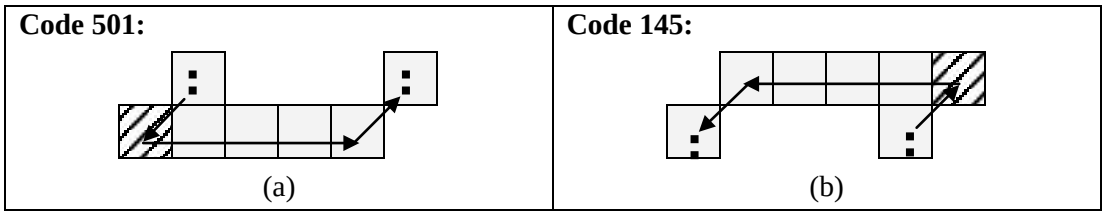


Fig. 7.20 – (a) Code number 501 (b) Code number 145

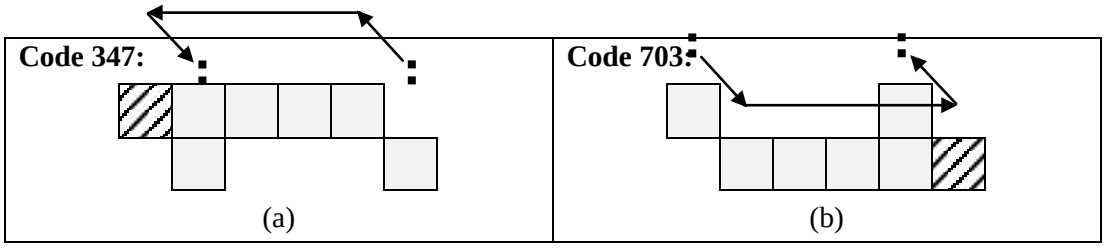


Fig. 7.21 – (a) Code number 347 (b) Code number 703

```
// The smoothing pseudo program for the code number 347. bc[i-3].direc ≠ 5 because
// code numbers 5347 is also possible.

if (bc[i].direc = 7 and bc[i-1].direc = 4 and  bc[i-2].direc = 3 and bc[i-3].direc ≠ 5)
{
    y1 ← bc[i].st_p_y;    x1 ← bc[i].st_p_x;
    output_Image (x1, y1) ← 0;
}
```

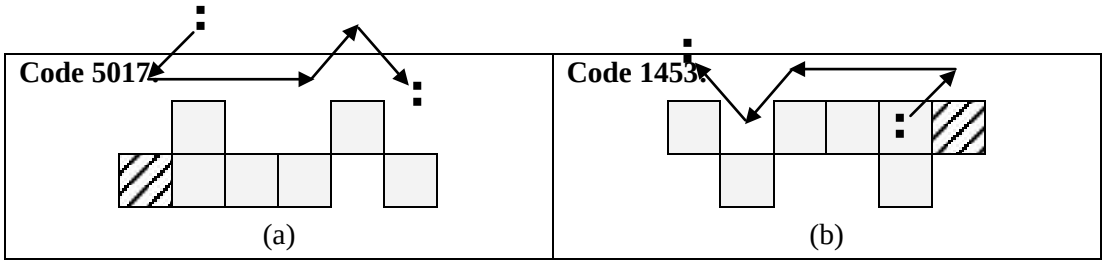


Fig. 7.22 – (a) Code number 5017 (b) Code number 1453

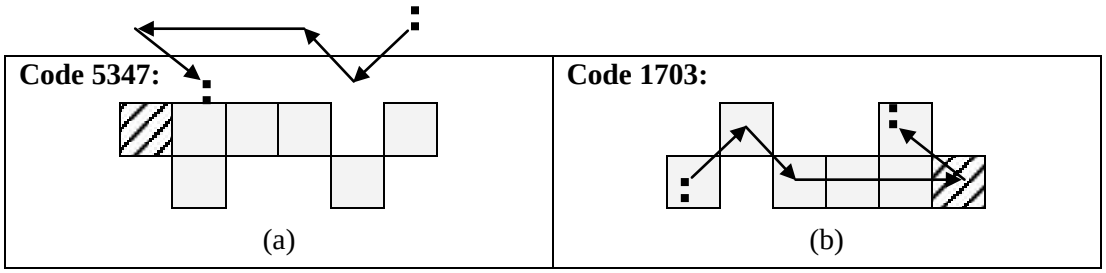


Fig. 7.23 – (a) Code number 5347 (b) Code number 1703

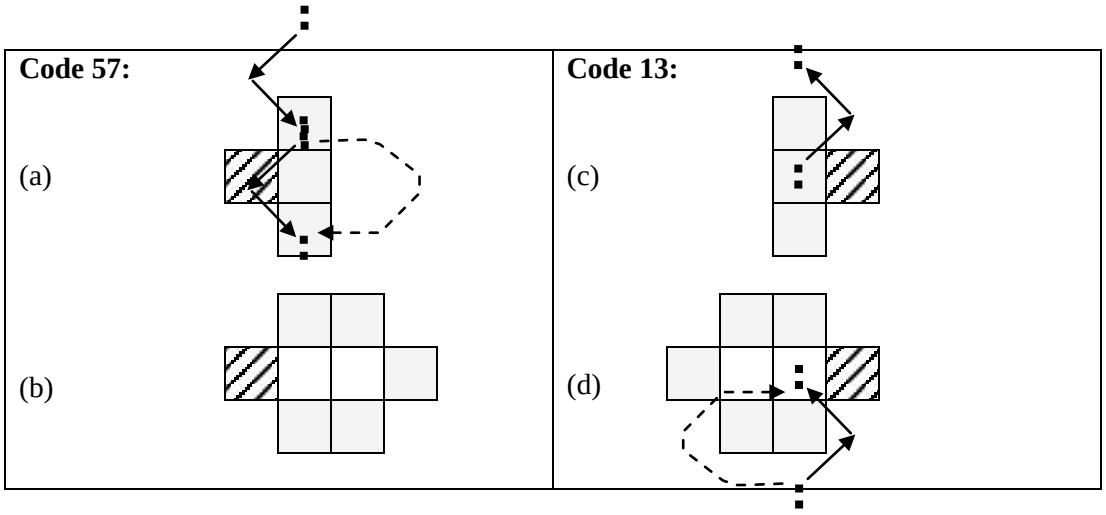


Fig. 7.24 – (a), (b) Code number 57 (c), (d) Code number 13. Smoothing pseudo programs similar to code numbers 54017 and 10453 should be written for code numbers 57 and 13. For example in case (a), the starting column of the out-spike's run is increased by one. However, in case (b), the out-spike's run is removed from the RLE data structure and then a new path on the boundary, shown by a dashed-line arrow, is traversed by the RLE contour tracer.

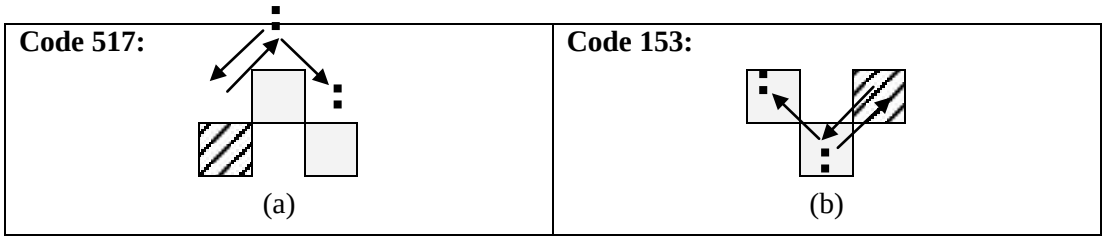


Fig. 7.25 – (a) Code number 517 (b) Code number 153

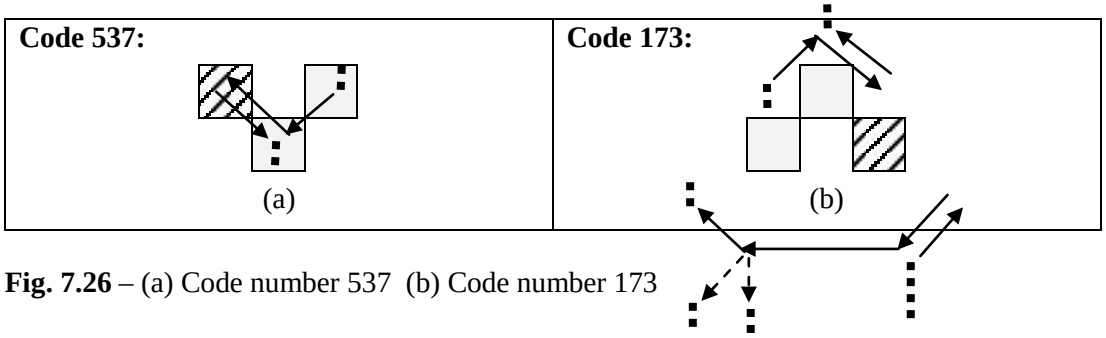


Fig. 7.26 – (a) Code number 537 (b) Code number 173

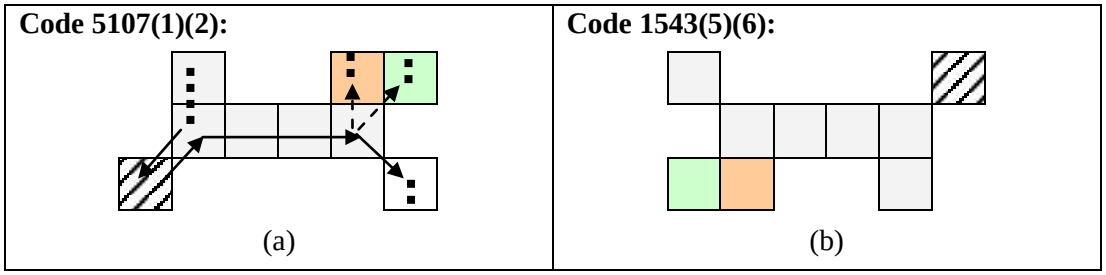


Fig. 7.27 – (a) Code number 5107(1)(2) (b) Code number 1543(5)(6)

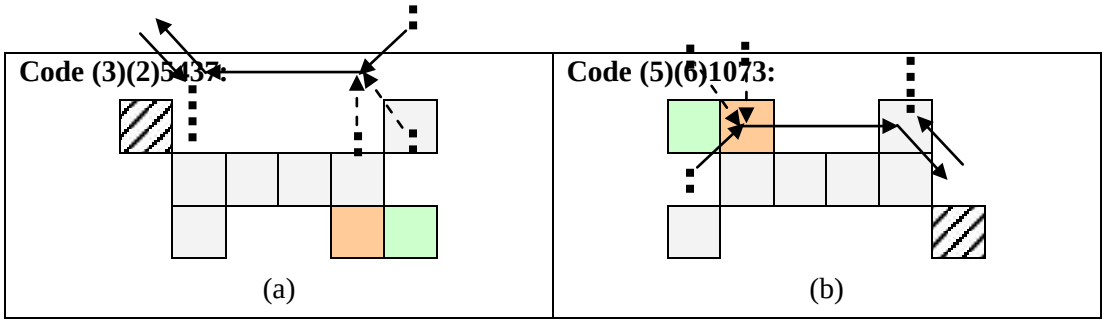


Fig. 7.28 – (a) Code number (3)(2)5437 (b) Code number (5)(6)1073

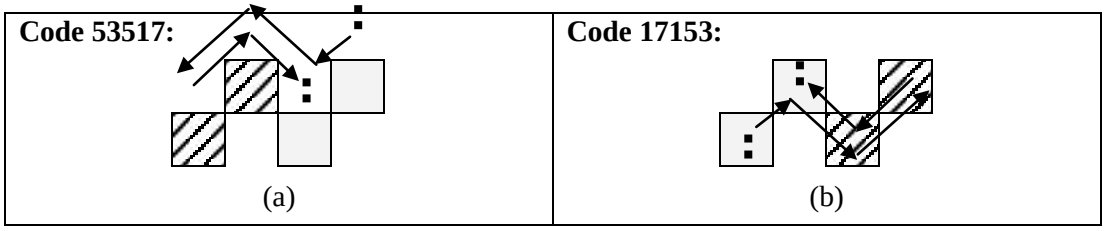


Fig. 7.29 – (a) Code number 53517 (b) Code number 17153

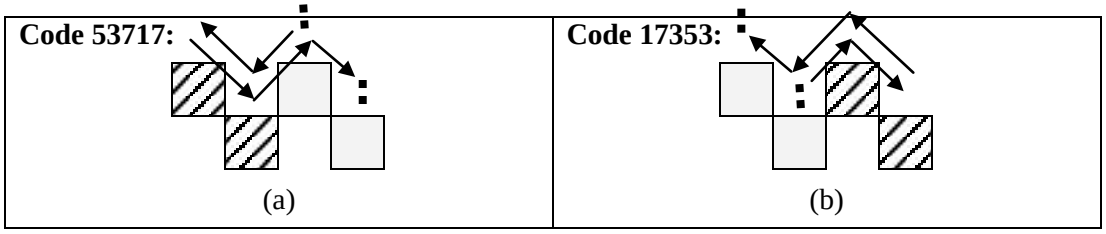


Fig. 7.30 – (a) Code number 53717 (b) Code number 17353

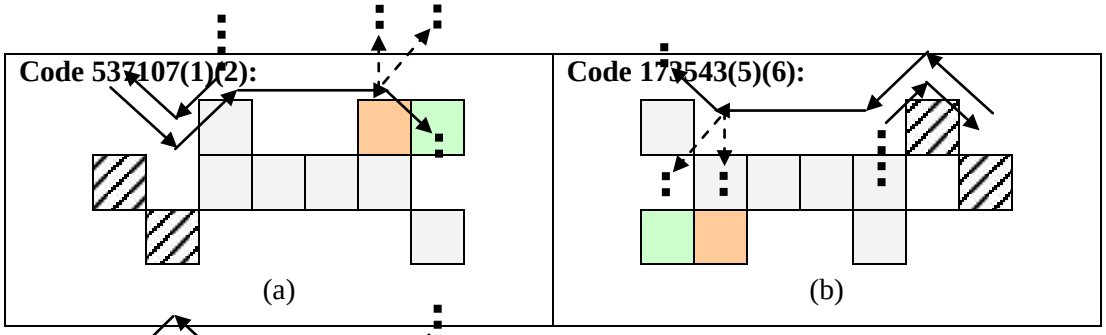


Fig. 7.31 – (a) Code number 537107(1)(2) (b) Code number 173543(5)(6)

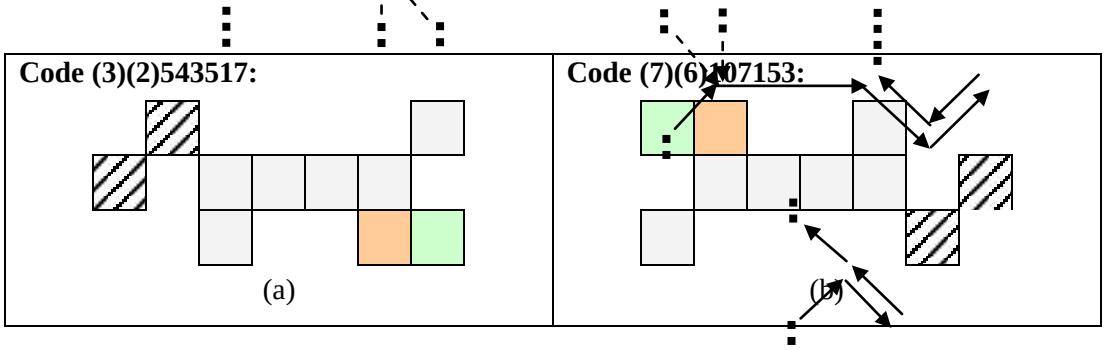


Fig. 7.32 – (a) Code number (3)(2)543517 (b) Code number (7)(6)107153

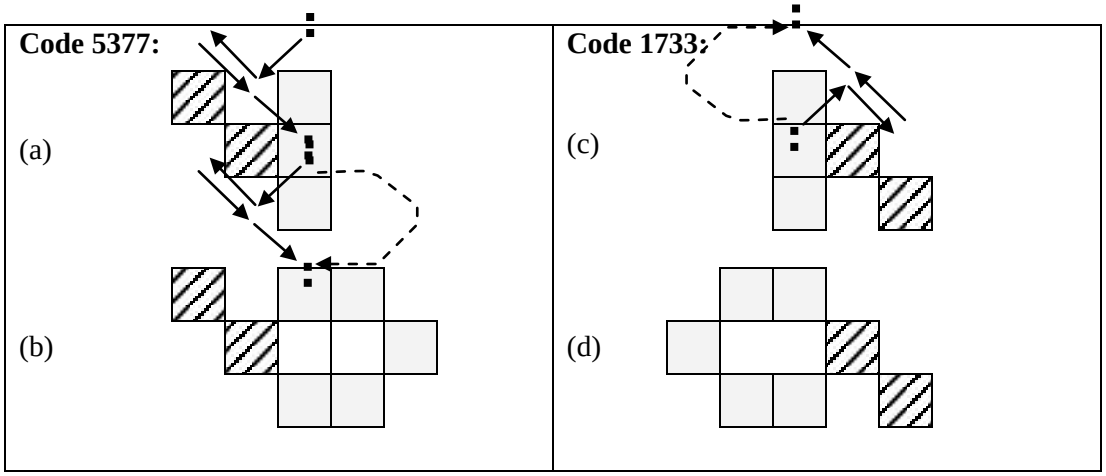


Fig. 7.33 – (a), (b) Code number 5377 (c), (d) Code number 1733. Smoothing pseudo programs similar to code numbers 57 and 15 should be written for code numbers 5377 and 1733, respectively.

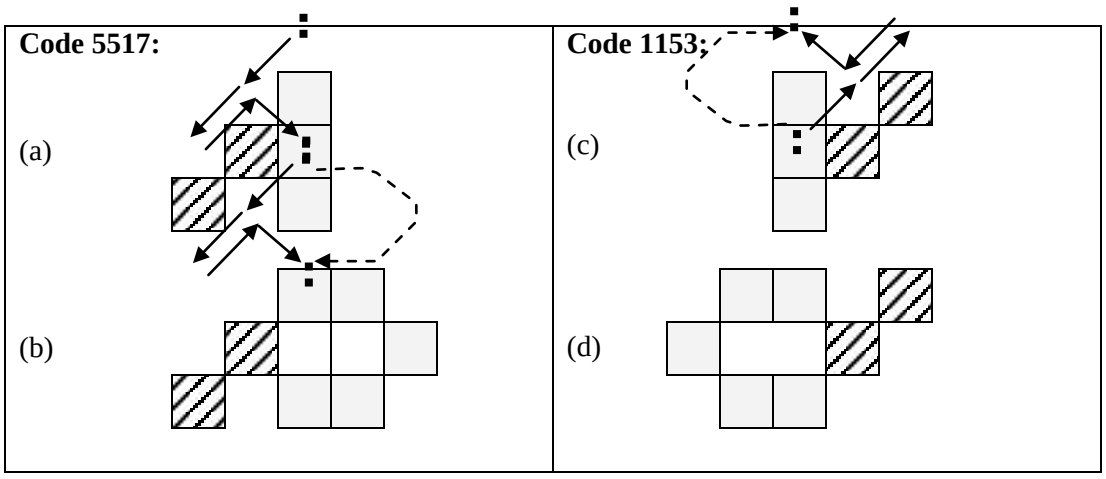


Fig. 7.34 – (a), (b) Code number 5517 (c), (d) Code number 1153.

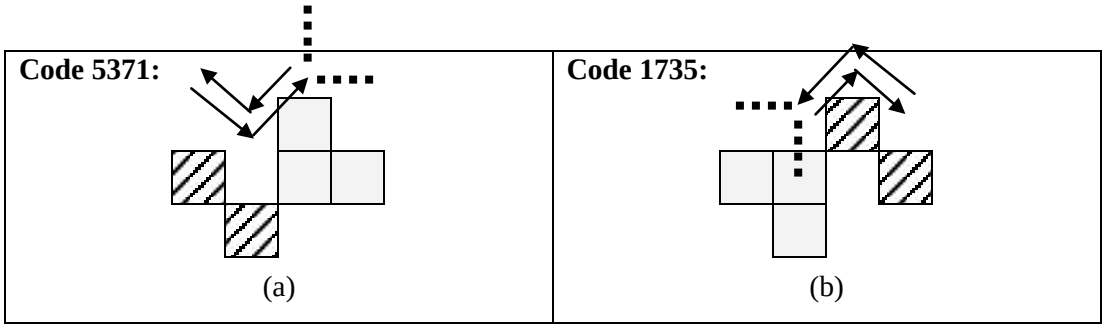


Fig. 7.35 – (a) Code number 5371 (b) Code number 1735

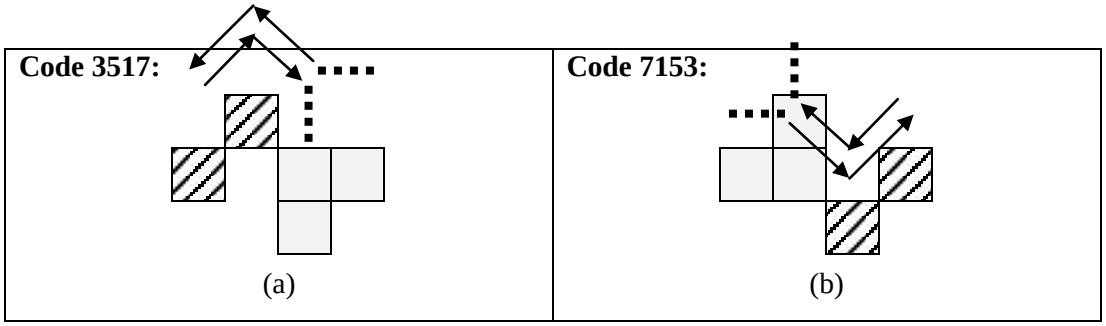


Fig. 7.36 – (a) Code number 3517 (b) Code number 7153

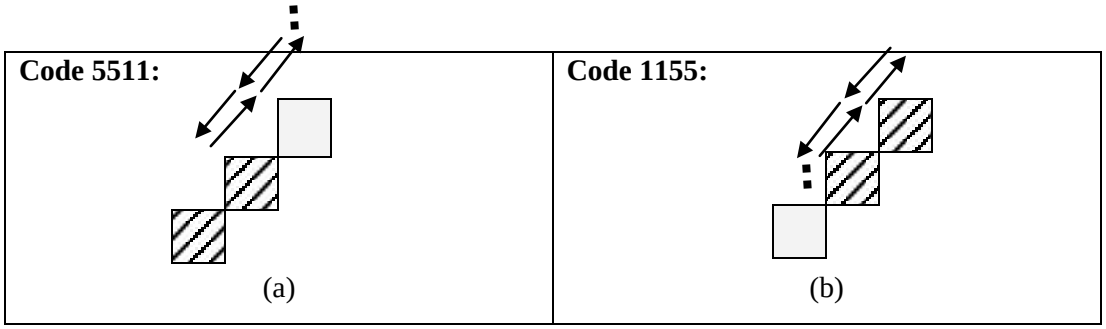


Fig. 7.37 – (a) Code number 5511 (b) Code number 1155

```
// The smoothing pseudo program for the code number 5511. bc[i].direc ≠ 7 and
//  bc[i].direc ≠ 0 because code numbers 55117 and 551107 are also possible.

if ((bc[i].direc ≠ 7 and bc[i].direc ≠ 0) and bc[i-1].direc = 1 and bc[i-2].direc = 1
     and bc[i-3].direc = 5 and bc[i-4].direc = 5)
{
     y1 ← bc[i-3].en_p_y;    x1 ← bc[i-3].en_p_x;
     output_Image [y1][x1] ← 0;

     y1 ← bc[i-4].en_p_y;    x1 ← bc[i-4].en_p_x;
     output_Image [y1][x1] ← 0;
}
```

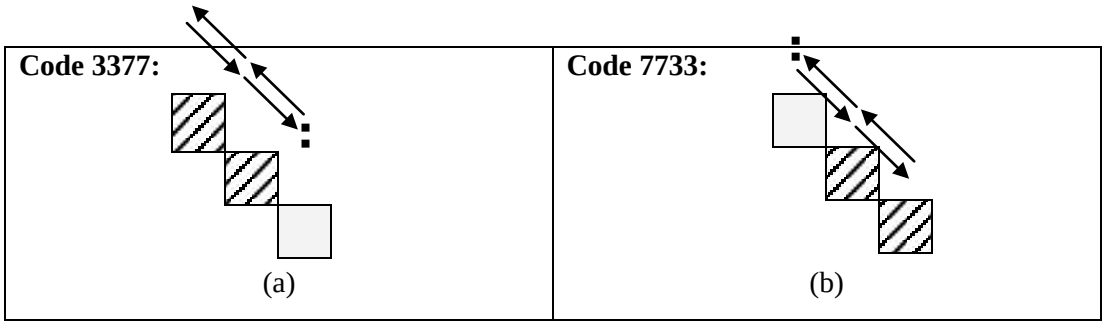


Fig. 7.38 – (a) Code number 3377 (b) Code number 7733

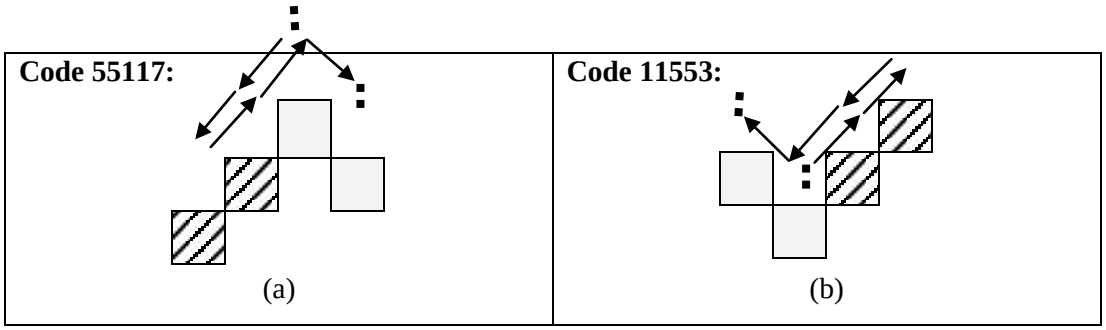


Fig. 7.39 – (a) Code number 55117 (b) Code number 11553

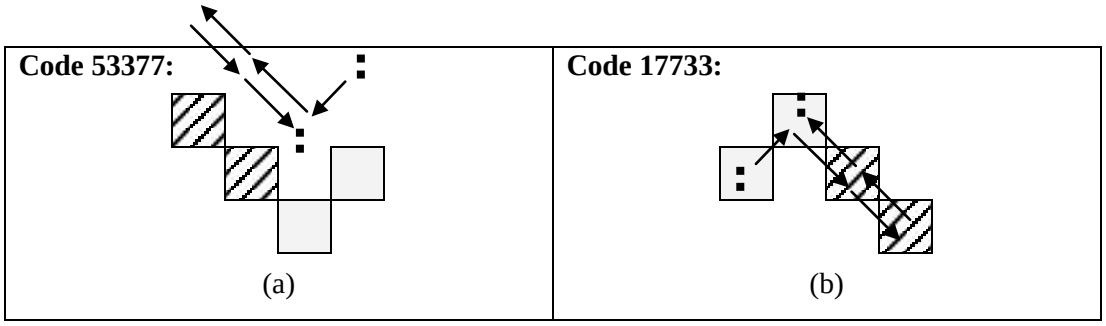


Fig. 7.40 – (a) Code number 53377 (b) Code number 17733

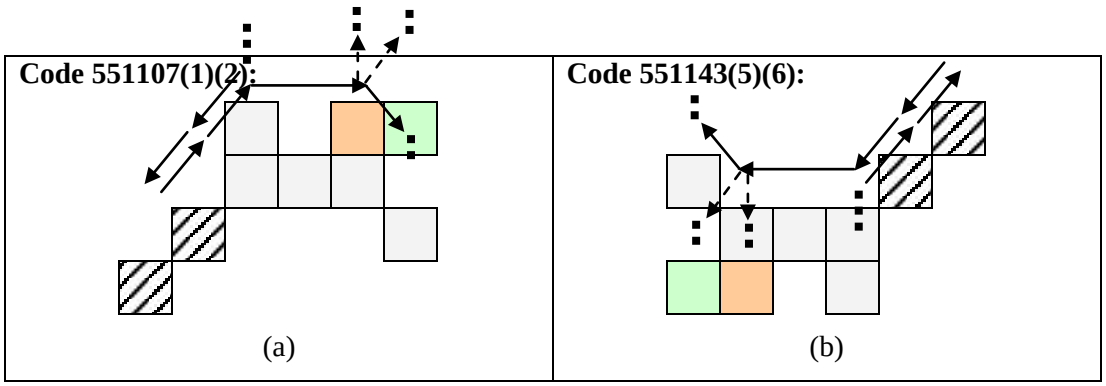


Fig. 7.41 – (a) Code number 551107(1)(2) (b) Code number 115543(5)(6)

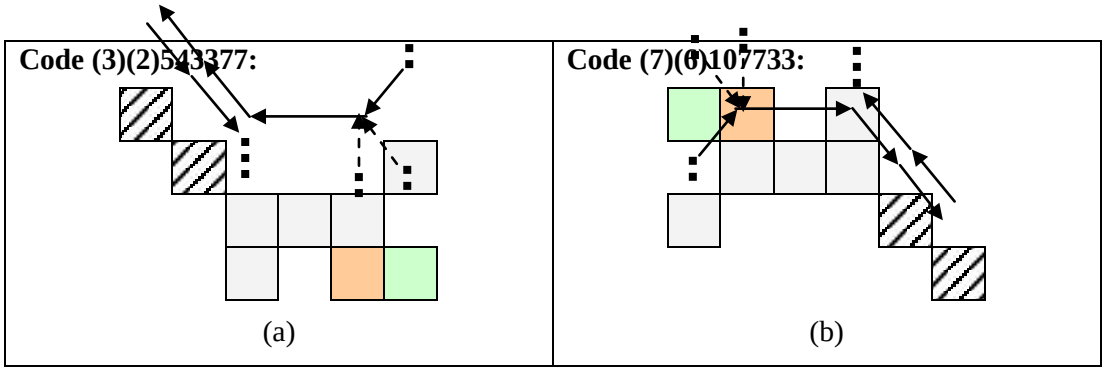


Fig. 7.42 – (a) Code number (3)(2)543377 (b) Code number (7)(6)107733

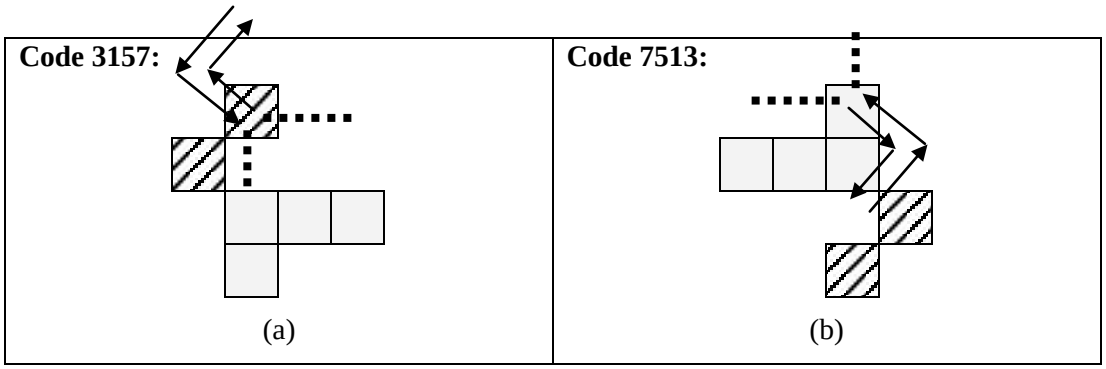


Fig. 7.43 – (a) Code number 3157 (b) Code number 7513

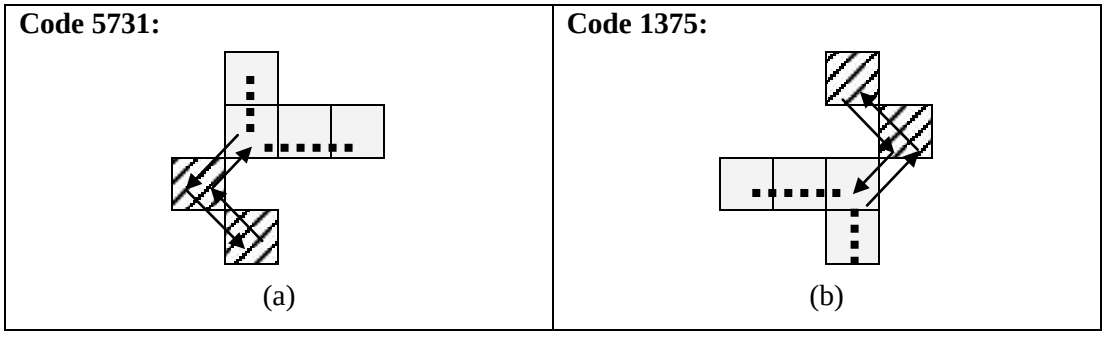


Fig. 7.44 – (a) Code number 5731 (b) Code number 1375

7.2.5 Top side and bottom side RLE boundary smoothing forms

Left side and right out-spike forms were offered in section 7.2.4. In this section, a number of forms for smoothing top side and bottom side out-spikes are proposed.

One major problem in recognising top side and bottom side out-spikes is that the data structure of a boundary code is incapable to describe these out-spikes effectively. To explain this problem, for example consider the top side code number 365(!3&!4) which is depicted in Fig. 7.45(a). As is observed in Fig. 7.45(a), it consists of two or more vertical pixels. The number of vertical pixels appearing in practice is unknown. The problem which arises is that how the data structure of a boundary code can describe different number of vertical pixels by only one code number? In other words, is it possible to recognise all code numbers 365, 3665, 36665, …, 36..65 etc only by code number 365?

Two approaches can be adopted to resolve the above problem:

1. Limit the number of vertical pixels to a maximum value, e.g. three or four pixels.

2. Select a more complete data structure.

The first approach is restrictive and greatly reduces the generality of the RLE smoothing method. A better approach is to find a solution using a data structure similar to the data structure of a boundary code. For this purpose, the previous data structure of a boundary code is extended by adding two extra fields called "*first_i*" and "*last_i*". Meanwhile, for simplicity of writing RLE smoothing programs, a boundary code and its extended form are shown by bc[i] and bc1[ii], respectively.

In the "$WALK_l$" and "$WALK_r$" functions, a bc1[ii]'s data structure is filled simultaneously, in a similar fashion as a bc[i]'s data structure. For all Freeman directions except 2 and 6, "*first_i*" and "*last_i*" have the same values and both are assigned with the index '*i*'. Once a vertical pixel is traversed by Quek's RLE boundary tracing algorithm, i.e. when the Freeman direction is 2 or 6, the indices '*i*' and '*ii*' are increased and "*first_i*" and "*last_i*" are assigned by index '*i*'. If there are more vertical pixels, the direction is not changed. In this case, the index '*i*' is increased and is assigned to "*last_i*". However, the index '*ii*' is not increased and "*first_i*" is not changed. In this case, if a Freeman direction except 2 or 6 occurs later, the indices '*i*' and '*ii*' are increased again and "*first_i*" and "*last_i*" are assigned by index '*i*'. Thus, the code numbers for top side and bottom side out-spikes, which contain a number of consecutive Freeman directions 2 and 6, are expressed by just one digit 2 or 6. As a result, regardless of the number of the vertical pixels, a top side or a bottom side out-spike form can be represented by a code number that consists of one digit 2 or 6 instead of a repetitive number of 2 or 6. Therefore, the left side and right side out-spikes are recognised by bc[i] boundary codes while the top side and bottom side out-spikes are recognised by bc1[ii] boundary codes.

In the following, the graphic forms representing the top side and bottom side out-spikes are illustrated in the left side tables, respectively. Their corresponding

converted forms (i.e. the bottom side and top side out-spikes) are also depicted in the right side tables. In addition, the out-spikes' pixels are shown as cross-hatched pixels and the smoothed RLE boundaries consist of only simple grey/green/brown pixels. Meanwhile, for obtaining the bottom side form of a top side out-spike such as code number 365(!3&!4) (Fig. 7.45(a)), i.e. code number 721(!7&!0), it is easily obtained by rotating the graphic form of code number 365(!3&!4) in counter clockwise direction by 180 degrees (Fig. 7.45(b)). These code numbers are suitable for smoothing the top left corner and bottom right corner of a foreground region. On the other hand, in order to obtain the smoothing form for the bottom left corner corresponding to code number 365(!3&!4), it is necessary to turn the graphic form of code number 365(!3&!4) upside down and then reverse its arrows' directions (Fig. 7.46(a)). The corresponding top right corner smoothing form is also obtained by either one of the above mentioned methods (Fig. 7.46(b)).

For recognising code number 365(!3&!4) in Fig. 7.45(a), four consecutive direction codes are checked where the rightmost digit should not be 3 and 4 because there are code numbers 3653, 3654(!3) and 36543 which separately recognise these special cases. Based on the graphic form 7.49(a), 365(!3&!4) is equivalent to code numbers 3650, 3651, 3652, 3655, 3656, 3657 while the code number 3652 is impossible to occur. If there are extra single-pixel out-spikes except for the cross-hatched one shown in Fig. 7.45(a), they are not smoothed at this stage. However, they are often recognised by other code numbers and are smoothed at later stages. Other code numbers such as 721(!7&!0) are checked in a similar manner.

The bottom left corner corresponding to code number 365(!3&!4) should be (!1&!0)761, as stated above. However, a direction code before a code number does not always show considerable information; since it is not important from what direction a code number is entered. The direction code(s) which usually occur(s) after a code number is more valuable. In this regard, the bottom left corner corresponding to code number 365(!3&!4) will be 761(!7&!0). In other words, in the upside down transformation of a code number followed by direction codes,

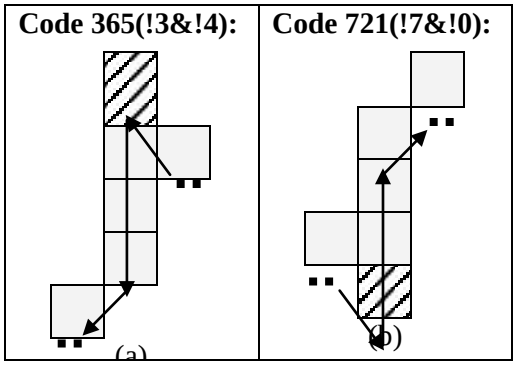


Fig. 7.45 – (a) Code number 365(!3 & !4) for top left corner; (b) Code number 721(!7 & !0) for bottom right corner.

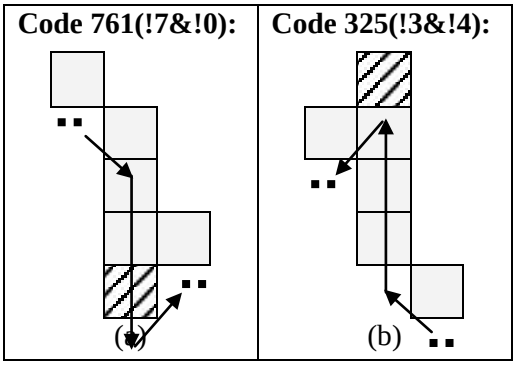


Fig. 7.46 – (a) Code number 761(!7 & !0) for bottom left corner; (b) Code number 325(!3 & !4) for top right corner.

converted versions of direction codes may only be maintained after the converted code number, but not before it. Based on this result, after the up side down transformation, regardless of whether a code number is followed by a combination of direction codes or not, the first direction code will be ignored. Then a direction code (or a combination of direction codes based on the "$WALK_l$" and "$WALK_r$" functions) is added to the converted code number.

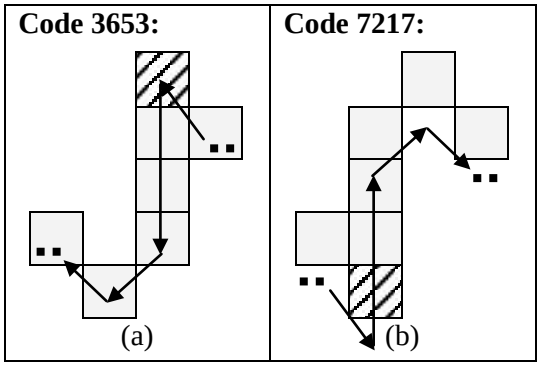


Fig. 7.47 – (a) Code number 3653
(b) Code number 7217

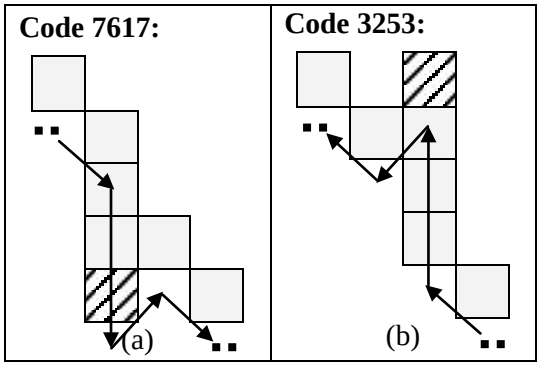


Fig. 7.48 – (a) Code number 7617
(b) Code number 3253

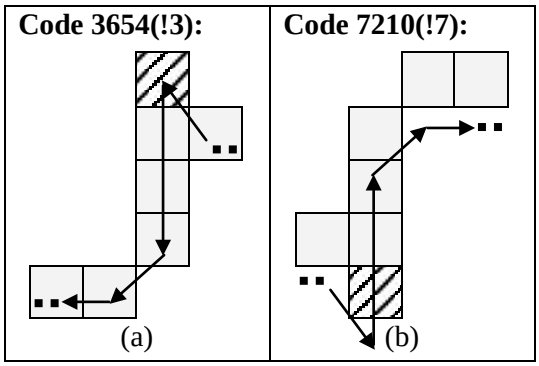


Fig. 7.49 – (a) Code number 3654(!3)
(b) Code number 7210(!7)

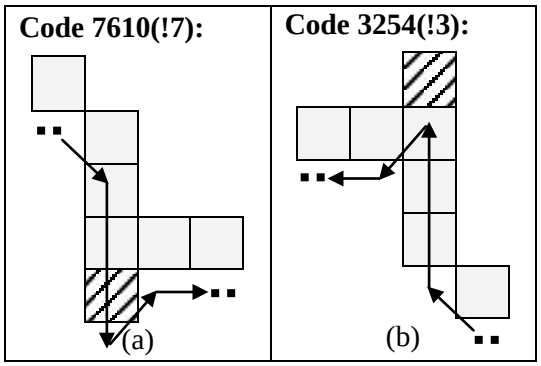


Fig. 7.50 – (a) Code number 7610(!7)
(b) Code number 3254(!3)

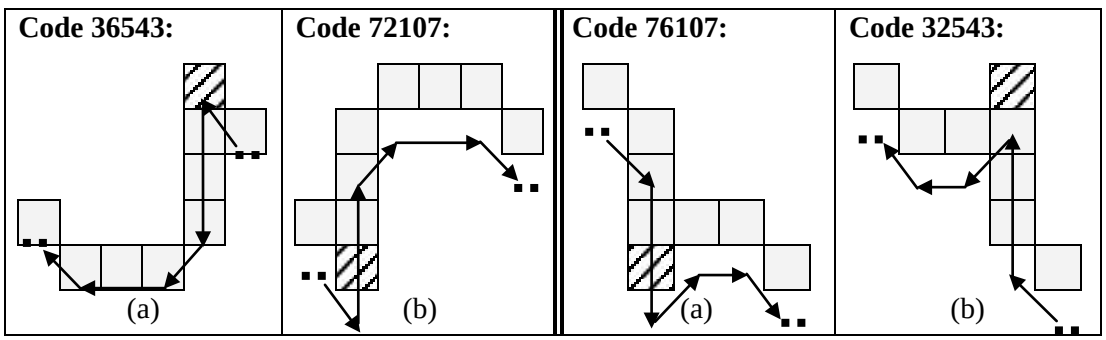


Fig. 7.51 – (a) Code number 36543
(b) Code number 72107

Fig. 7.52 – (a) Code number 76107
(b) Code number 32543

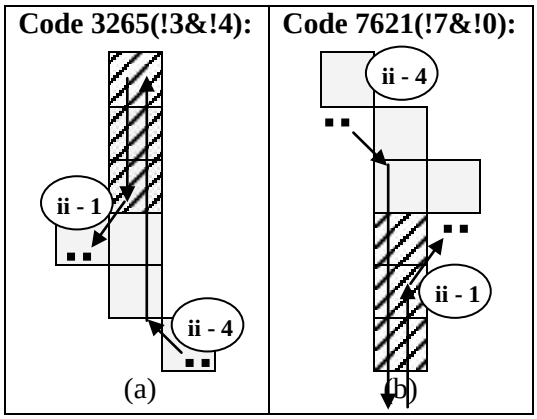


Fig. 7.53 – (a) Code number 3265(!3&!4):
bc1[ii-1].en_p_y < bc1[ii-4].st_p_y
(b) Code number 7621(!7&!0):
bc1[ii-1].en_p_y≥ bc1[ii-4].st_p_y

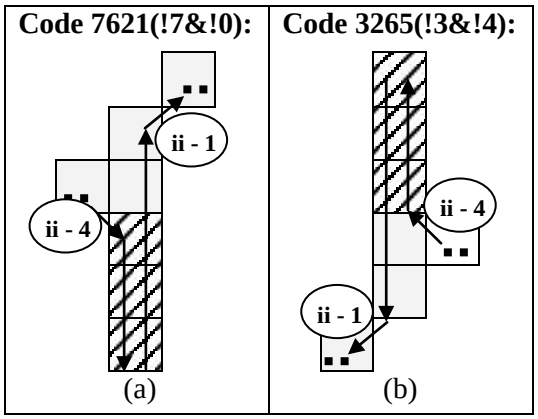


Fig. 7.54 – (a) Code number 7621(!7&1!0):
bc1[ii-1].en_p_y < bc1[ii-4].st_p_y
(b) Code number 3265(!3&!4):
bc1[ii-1].en_p_y≥ bc1[ii-4].st_p_y

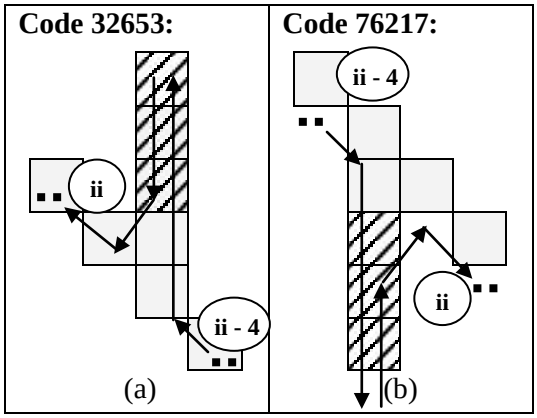


Fig. 7.55 – (a) Code number 32653:
bc1[ii].en_p_y < bc1[ii-4].st_p_y
(b) Code number 76217:
bc1[ii].en_p_y≥ bc1[ii-4].st_p_y

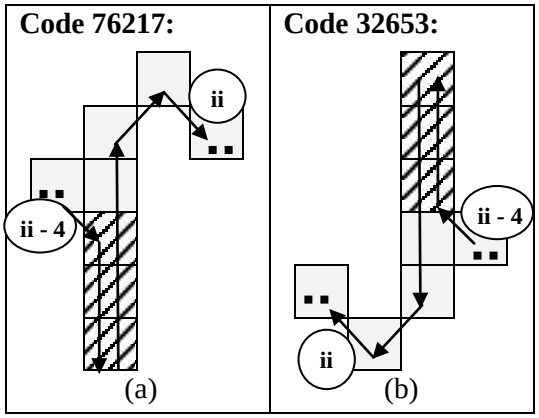


Fig. 7.56 – (a) Code number 76217:
bc1[ii].en_p_y < bc1[ii-4].st_p_y
(b) Code number 32653:
bc1[ii].en_p_y≥ bc1[ii-4].st_p_y

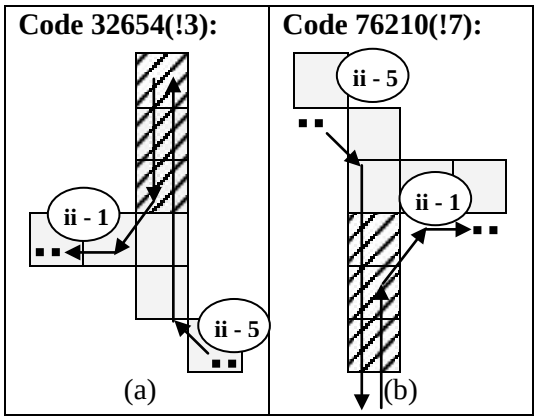


Fig. 7.57 – (a) Code number 32654(!3):
bc1[ii-1].en_p_y < bc1[ii-5].st_p_y
(b) Code number 76210(!7):
bc1[ii-1].en_p_y≥ bc1[ii-5].st_p_y

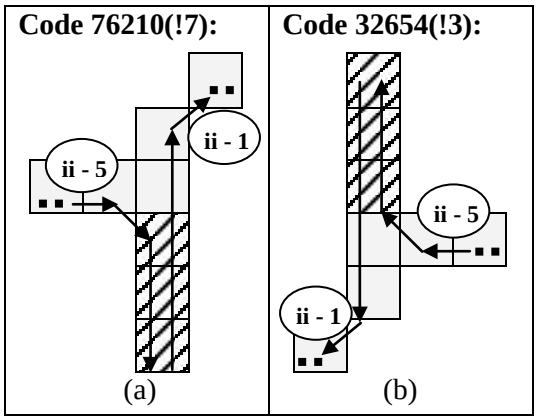


Fig. 7.58 – (a) Code number 76210(!7):
bc1[ii-1].en_p_y < bc1[ii-5].st_p_y
(b) Code number 32654(!3):
bc1[ii-1].en_p_y≥ bc1[ii-5].st_p_y

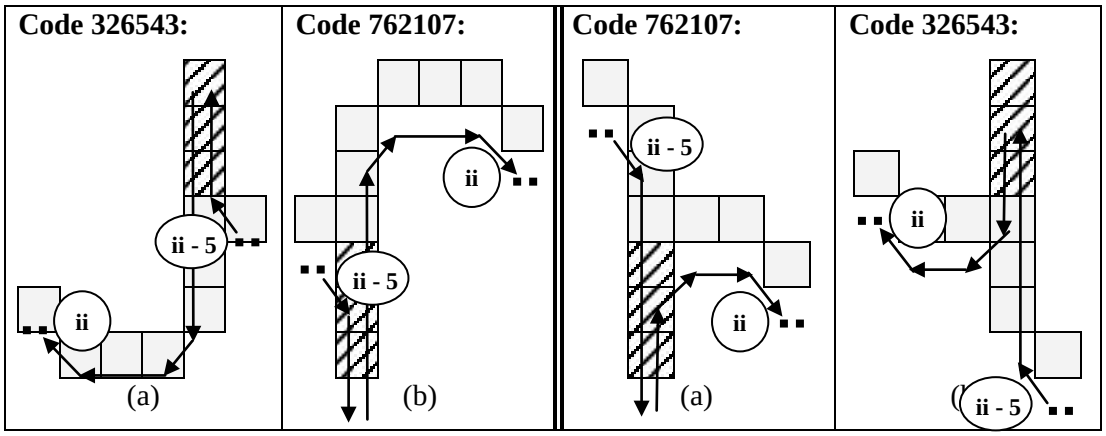


Fig. 7.59 – (a) Code number 326543:
bc1[ii].en_p_y > bc1[ii-5].st_p_y
(b) Code number 762107:
bc1[ii].en_p_y ≤ bc1[ii-5].st_p_y

Fig. 7.60 – (a) Code number 762107:
bc1[ii].en_p_y > bc1[ii-5].st_p_y
(b) Code number 326543:
bc1[ii].en_p_y ≤ bc1[ii-5].st_p_y

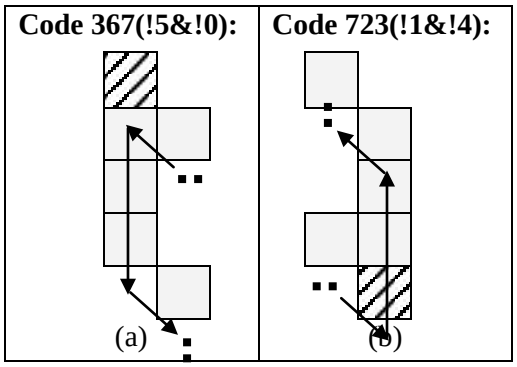


Fig. 7.61 – (a) Code number 367(!5&!0)
(b) Code number 723(!1&!4)

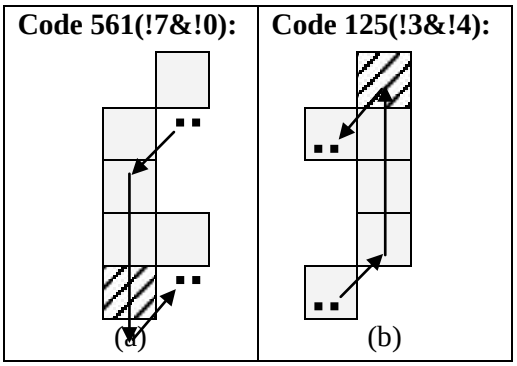


Fig. 7.62 – (a) Code number 561(!7&!0)
(b) Code number 125(!3&!4)

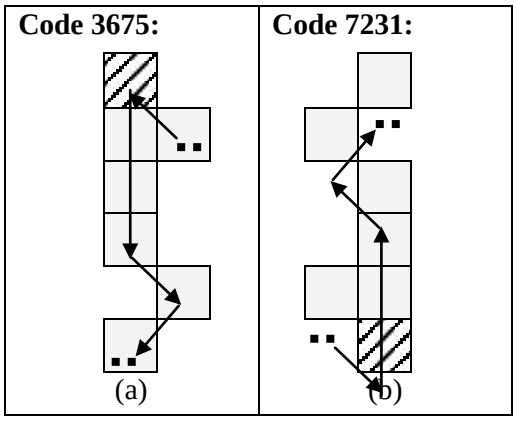


Fig. 7.63 – (a) Code number 3675
(b) Code number 7231

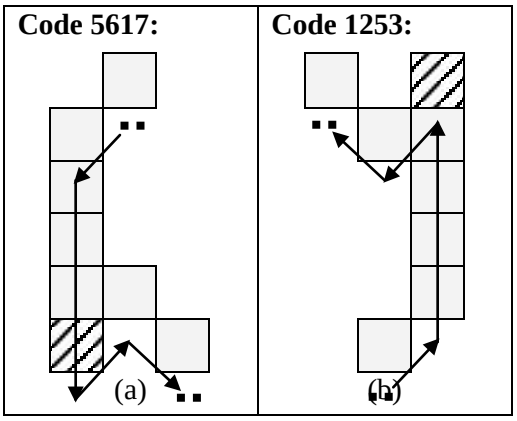


Fig. 7.64 – (a) Code number 5617
(b) Code number 1253

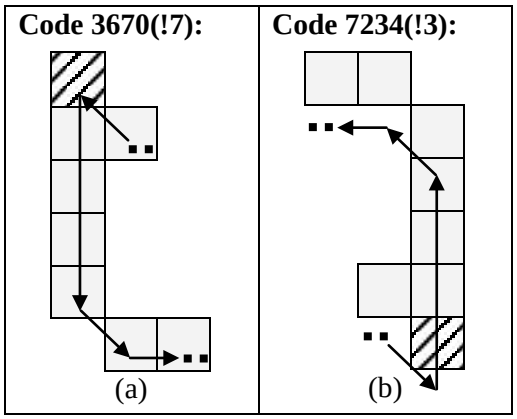


Fig. 7.65 – (a) Code number 3670(!7)
(b) Code number 7234(!3)

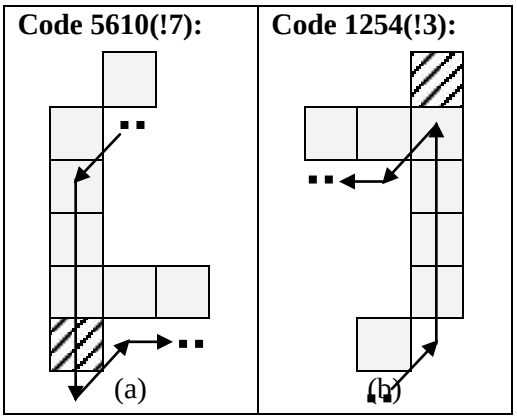


Fig. 7.66 – (a) Code number 5610(!7)
(b) Code number 1254(!3)

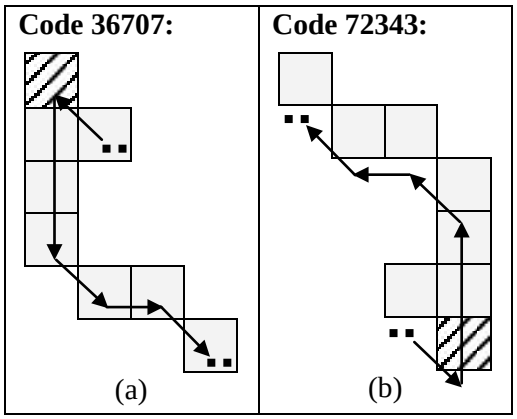


Fig. 7.67 – (a) Code number 36707
(b) Code number 72343

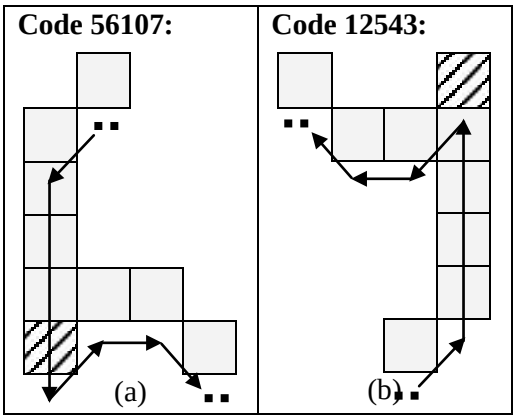


Fig. 7.68 – (a) Code number 56107
(b) Code number 12543

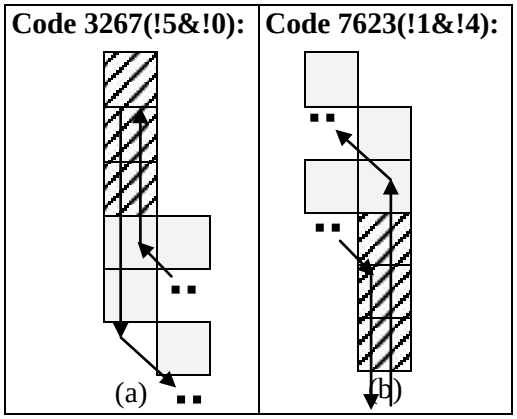


Fig. 7.69 – (a) Code number 3267(!5&!0)
(b) Code number 7623(!1&!4)

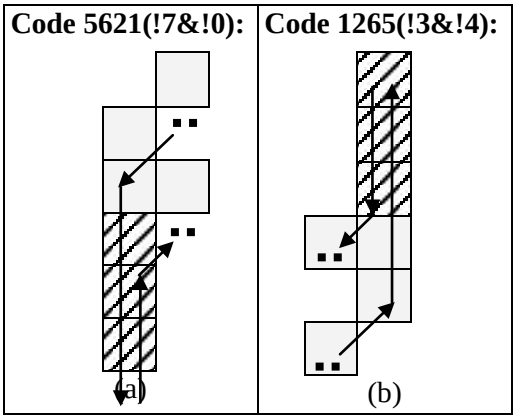


Fig. 7.70 – (a) Code number 5621(!7&!0)
(b) Code number 1265(!3&!4)

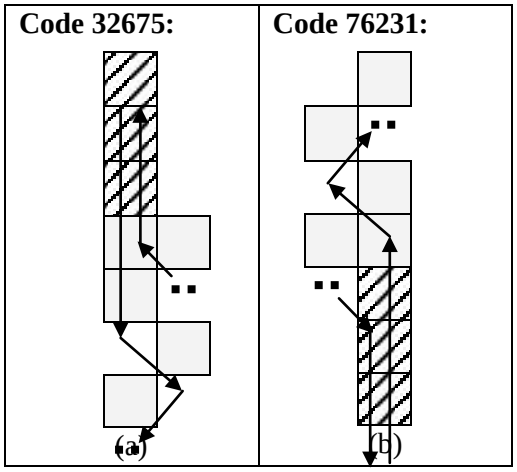


Fig. 7.71 – (a) Code number 32675
(b) Code number 76231

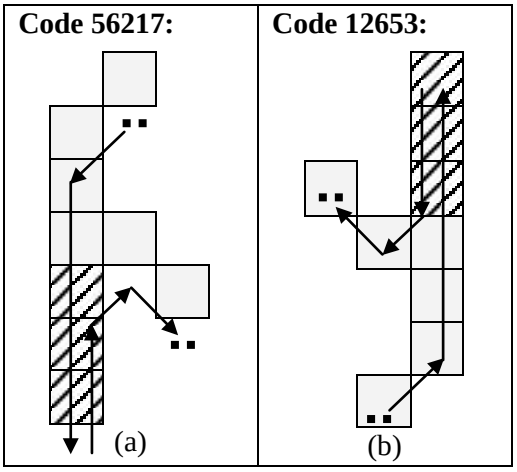


Fig. 7.72 – (a) Code number 56217
(b) Code number 12653

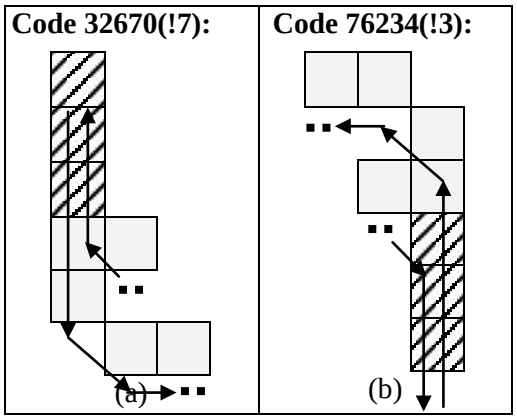


Fig. 7.73 – (a) Code number 32670(!7)
(b) Code number 76234(!3)

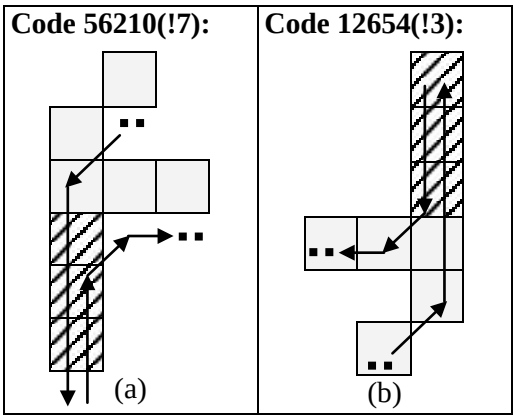


Fig. 7.74 – (a) Code number 56210(!7)
(b) Code number 12654(!3)

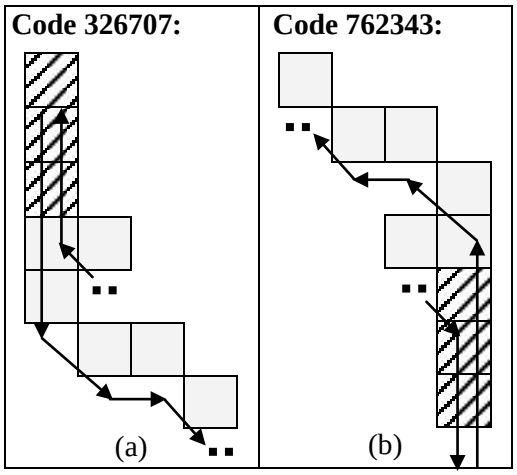


Fig. 7.75 – (a) Code number 326707
(b) Code number 762343

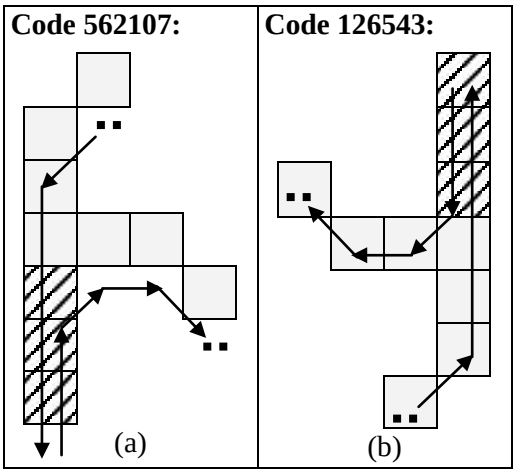


Fig. 7.76 – (a) Code number 562107
(b) Code number 126543

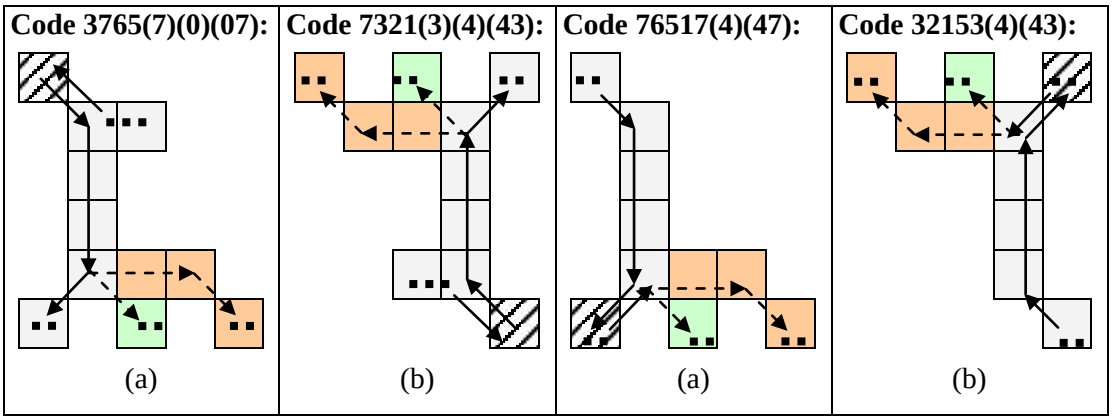


Fig. 7.77 – (a) Code number 3765(7)(0)(07)
(b) Code number 7321(3)(4)(43)

Fig. 7.78 – (a) Code number 76517(4)(47)
(b) Code number 32153(4)(43)

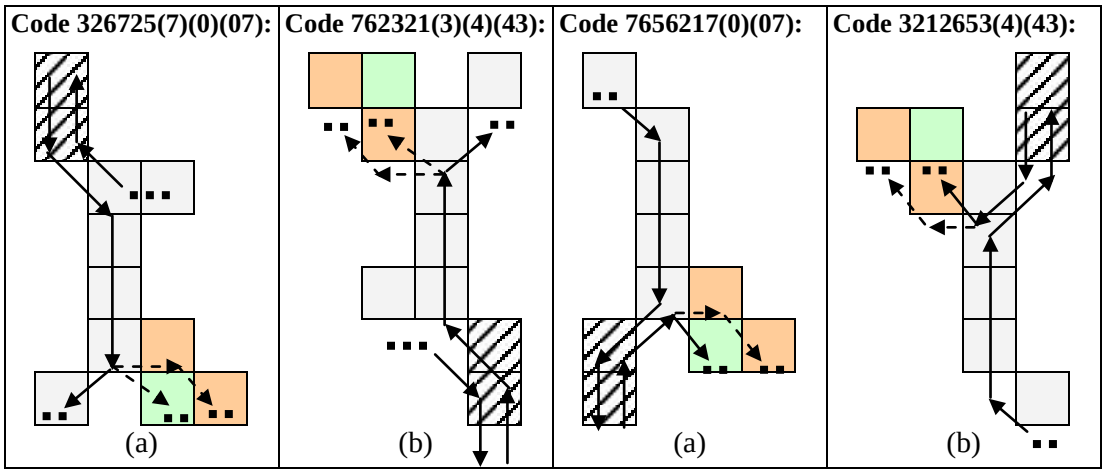


Fig. 7.79 – (a) Code number 326725(7)(0)(07)
(b) Code number 762321(3)(4)(43)

Fig. 7.80 – (a) Code number 7656217(0)(07)
(b) Code number 3212653(4)(43)

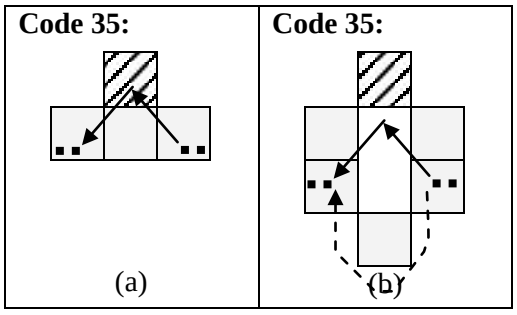


Fig. 7.81 – (a), (b) Code number 35

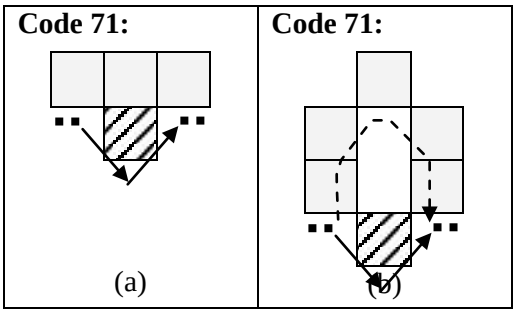


Fig. 7.82 – (a), (b) Code number 71

Although a large number of RLE smoothing forms have been offered, there are still a number of other forms, which for the sake of brevity, have not been depicted. All the above forms and some other ones have been included in the RLE smoothing software.

7.2.6 The strategies of applying the RLE smoothing forms

The RLE smoothing forms required for removing left side, right side, top side and bottom side out-spikes were given in sections 7.2.4. and 7.2.5, respectively. However, the strategies of applying these RLE smoothing forms are as important as the smoothing forms. Three strategies are outlined as follows:

1. Once Quek's boundary tracing algorithm begins from the starting point of a region boundary, i.e. the top-leftmost unused end of a foreground region's run, the RLE min-of-two smoothing method is called after each "$WALK_l$" and "$WALK_r$" function. Suppose there are two nested out-spikes such that when the second out-spike is recognised and smoothed sooner, the first out-spike will appear afterwards. However, using this strategy, only the second out-spike is smoothed and the first one remains unchanged. In this case, RLE boundary tracing and smoothing are not finished by reaching to the starting point of the region. Instead they continue their processes unless no further smoothing is possible. Based on this strategy, a region boundary may be traced a number of times so that all out-spikes are removed. Thus, this is obviously a time-consuming and ineffective strategy.

2. After applying a smoothing form to an out-spike, usually one or more runs in the RLE data structure are modified (or removed) and one or more pixels are deleted from the output image. Besides, in the second strategy, bc[i]'s and bc1[ii]'s data structures should also be re-organised for each code number in the smoothing function. The goal of re-organising bc[i]'s and bc1[ii]'s data structures is to show the effect of the removed out-spike(s) in that region segment. However, this strategy is still not efficient enough due to the following reasons:

   - For re-organising, bc[i]'s and bc1[ii]'s data structures corresponding to the removed pixels should also be omitted. In this case, bc[i]'s and bc1[ii]'s data structures corresponding to pixels occurring after the removed pixels should be shifted back. Thus the programming code for each code number becomes more difficult and will take a long time for the programmer to write the whole smoothing program. In addition, shifting a number of bc[i]'s and bc1[ii]'s data structures at the run-time can be a time-consuming process. This is especially a lengthy process when only one noisy pixel occurring at the beginning of a code number should be removed. In this case, more numbers of the remained bc[i]'s and bc1[ii]'s data structures should be shifted back.

   - Suppose there are two nested out-spikes such that when initially the second out-spike is recognised and smoothed, the first out-spike will appear as a result. However, after the re-organising process since Quek's boundary tracing algorithm continues, the first out-spike remains unrecognised. Thus, there is no guarantee that smoothing is finished only by a single traversal of a region boundary. As a result, similar to strategy one, re-organising strategy should again be repeated for a number of times so that all the out-spikes to be removed from a region boundary.

3. In the third strategy, at first, all the pixels of an out-spike are removed from the output image and the runs in the RLE data structure are modified (or removed). Then the tracing point of Quek's algorithm is backtracked to the starting pixel of the current out-spike form before smoothing. For implementing the backtracking strategy, all necessary input parameters passed to the "$WALK_l$" and "$WALK_r$" functions are stored in bc[i]'s and bc1[ii]'s data structures. This is performed immediately after entering these functions and before they traverse that segment of the region. Then the smoothing function is called. In the smoothing function, once a special code number matches with the out-spike form and the smoothing operation is performed, indices '*i*' and '*ii*' are reduced by the number of digits of that code number. As a result, Quek's contour tracer resumes its traversal again from the current boundary segment while in this case, there is no out-spike in that segment. In this way, when there are nested out-spikes, by smoothing the second out-spike, the first one will appear later. However, at this time, the first out-spike is recognised and smoothed by Quek's and the min-of-two methods, respectively. Therefore, in the backtracking strategy, only the values of indices '*i*' and '*ii*' are reduced by the number of digits checked in the smoothing function. Meanwhile, the state of the contour tracer is restored back to its previous state before the smoothing. Thus, no shifting of bc[i]'s and bc1[ii]'s data structures will be required and it will be a fast approach.

The implementation of the backtracking strategy includes storing the necessary input parameters of the "$WALK_l$" (or "$WALK_r$") function and reducing indices '*i*' and '*ii*'. As an example, consider a pseudo code for the code number 5407 as follows:

```
// The necessary input parameters of the "WALKl" function, for example in state 4,
// are stored in bc[i]'s and bc1[ii]'s data structures before the boundary traversal is
// performed in this function (the input parameters of the "WALKr" function should
// be stored as well). These operations should be done for all states 1 to 8.

if (end_point_col = p_col – 1)  // state (1,4)
{
    i ← i +1;  ii ← ii +1;
    bc1[ii].direc ← 3;  bc[i].direc ← 3;
    bc1[ii].st_p_x ← p_col;  bc[i].st_p_x ← p_col;
    bc1[ii].st_p_y ← p_row;  bc[i].st_p_y ← p_row;
    bc1[ii].en_p_x ← end_point_col;  bc[i].en_p_x ← end_point_col;
    bc1[ii].en_p_y ← end_point_row;  bc[i].en_p_y ← end_point_row;
    bc1[ii].st_run ← p_run;  bc[i].st_run ← p_run;
    bc1[ii].en_run ← end_point_run;  bc[i].en_run ← end_point_run;
    bc1[ii].last_i ← i;  bc1[ii].first_i ← i;
    bc1[ii].cur_run ← end_run;  bc[i].cur_run ← end_run;
    bc[i].i_val ← i;
    bc1[ii].i_val ← ii;
    bc1[ii].next_st ← UP;  bc[i].next_st ← UP;
    bc1[ii].m ← m;  bc[i].m ← m;
}
```

After "$WALK_l$" (or "$WALK_r$"), the min-of-two smoothing method (function) is called and suppose an out-spike form matches with code number 5407 as follows:

```
// Inside min-of-two smoothing method
smooth_flag ← 0;
•
•
if (bc[i].direc = 7 and bc[i-1].direc = 0 and bc[i-2].direc = 4 and bc[i-3].direc = 5)
{
   // The body of smoothing pseudo code given on page 125 goes here

   i ← bc[i-3].i_val;  ii ← bc1[ii-3].i_val;
   smooth_flag ← 1;
}
•
•
// At the end of min-of-two smoothing method
if (smooth_flag = 1)
{
    p_col ← bc[i].st_p_x;
    p_row ← bc[i].st_p_y;
    cur_run ← bc[i].st_run;
    state ← bc[i].next_st;
    m ← bc[i].m;
 }
```

Then Quek's contour tracer resumes its traversal again from the current boundary segment. However, at this time there will be no out-spike in that segment. Therefore, Quek's algorithm extracts all boundary pixels by a single traversal of a region boundary as if there were no out-spike(s) around that foreground region at all.

Finally, even if the re-organising strategy is combined with the backtracking approach, it will still be ineffective. This is due to shifting back a large number of the remained bc[i]'s and bc1[ii]'s data structures at the run-time. Besides, this strategy requires a longer program for any code number than its corresponding program for the third strategy.

7.2.7 The min-of-two versus related methods

Before comparing the min-of-two method with related algorithms, let's consider the advantages and disadvantages of this method. The min-of-two smoothing method has the following advantages:

- The major advantage of the min-of-two multiple-point smoothing method is that it is complement of an effective RLE boundary tracing method (Quek's algorithm) and operates simultaneously with it. As a result, by applying the combination of Quek's and the min-of-two algorithms to each foreground binary region, its smoothed boundary pixels are extracted effectively.

- It is relatively a simple method because it is understandable and can be implemented in a straightforward manner.

- It is very fast due to a few numbers of tests and processing steps for each out-spike form without requiring any floating point operation.

- If there are n ($n \geq 2$) consecutive and nested out-spikes so that when the n-th out-spike is smoothed, the $n-1$-th out-spike appears later, then the min-of-two method has the capability to smooth $n-1$-th, and then $n-2$-th, … out-spikes, respectively (i.e. based on their order of appearances).

- The min-of-two smoothing method can remove all one-pixel-width out-spikes by a single traversal of a region boundary.

- It operates directly on RLE data. The RLE data structure is used for storing the information of foreground binary regions and also utilised by a number of blocks in this thesis.

- Similar to Quek's algorithm, it can smooth both external and internal region boundaries.

- After the smoothing operation, the amount of noise in boundary contours is reduced. Therefore, in comparison with morphological operations, this method not only smoothes the boundaries of foreground binary regions but it can also preserve their shapes with no or very little distortion.

However, the min-of-two smoothing method has the following disadvantage:

- A large number of forms have been proposed to remove noisy out-spikes. Therefore, it is more desirable if many similar patterns can be combined into more general patterns so that a shorter source code can achieve the same actions with a greater effectivity.

7.2.7.1 Comparison with chain code smoothing methods

The output of the min-of-two smoothing method may be compared with the output of chain code algorithms such as Suen et al. (1992), Yu and Yan (1997), and Hu et al., (1998) on the same number of binary images. For performing this comparison, first a contour tracing algorithm such as Ren et al. (2002) should be used to obtain the chain contours of foreground binary regions. Then all the above smoothing methods should also be implemented. However, since there are no defined qualitative or quantitative rules for the amount of smoothing of binary regions, no specific comparison can be made among different contour smoothing methods. Nevertheless, an approximate comparison between the min-of-two method with Yu and Yan (1997), and Hu et al., (1998) is given as follows:

- Both Yu and Yan's and Hu et al.'s algorithms require that a contour tracing method has already obtained the boundaries of the foreground binary regions.

However, the min-of-two method does not have such prerequisite since it operates simultaneously with Quek's RLE boundary tracing algorithm.

- Yu and Yan's algorithm is a single point smoothing approach but the min-of-two method can smooth both single and multiple point out-spikes.

- Although Hu et al.'s and the min-of-two algorithms are both multiple-point smoothing methods, Hu et al. utilise patterns which are designed for binary contour smoothing using chain codes. On the other hand, the min-of-two method is designed to apply patterns suitable for smoothing foreground regions represented by the RLE approach. In addition, it seems that Hu et al.'s patterns are more suitable for smoothing the images of handwritten numerals. However, there is no report of applying their method to images of foreground regions in other fields such as, for example, moving objects segmentation, etc. One possible reason for this may be too much smoothing and distortion or inadequacy of their patterns for foreground regions in such applications. However, the min-of-two smoothing method is a conservative one-pixel-width approach. It operates such that minimum distortion on the shapes of foreground objects occurs. It is more suitable for smoothing foreground binary regions shown using RLE representation in applications such as moving objects detection, visual surveillance, etc. Thus, these two methods may not correctly be compared with each other. This is because they are designed for different types of representations of foreground regions and different types of applications.

In addition to the above algorithms, let's compare Chan and Hsu's (2008) algorithm with the min-of-two method as follows:

- Chan and Hsu's algorithm, which operates on grey-scale images, uses two synchronously-running modules PBL and SRP. The PBL module traces the object's border pixel-by-pixel using a 3 x 3 sliding window while the SRP module removes random noisy pixels of the object's contour. Thus, similar to Quek's and the min-of-two algorithms, the PBL and the SRP modules play the role of boundary tracing and smoothing methods. Chan and Hsu's algorithm can be compared with the combination of Quek's algorithm and the min-of-two method as follows:

  - The decisions of the BPL and the SRP modules are based on the grey-scale differences between the centre pixel and its surrounding pixels in a 3 x 3 window. However, Quek and min-of-two methods operate on binary images. Thus, the SRP module and the min-of-two method are not comparable due to performing their processes on different types of images, i.e. grey-scale and binary images, respectively.

  - Since the boundary pixels of an object are centred in 3 x 3 sliding window used by the BPL module along the tracing direction, the width of noisy pixels to be removed by the SRP module is normally ranged from 1 to 3 pixels. As a result, the SRP module can be utilised for limited smoothing applications. However, the min-of-two method has more capability as it can

remove simple and nested one-pixel-width horizontal, vertical and diagonal out-spikes of any length.

- Chan and Hsu's algorithm is designed to work with chain code contours but has not been designed to operate on foreground binary regions represented by an RLE structure.

In fact, the BPL and the SRP modules are used for a different purpose, i.e. as a segmentation tool for grey-scale images (although are utilised for similar operations), while Quek and the min-of-two methods are utilised to obtain smoothed boundaries of foreground regions in binary images.

Fig. 7.83 shows three grey-scale images along with extracted contours using two standard methods and Chan and Hsu's algorithm (PSCTM). It is observed from the "actual object mask" of this figure that the contours of objects are almost smoothed since a few numbers of outliers are visible around the boundaries of those objects. Thus, a soft and limited smoothing method such as PSCTM is suitable in such cases. However, Figs. 7.84 to 7.86 show that the min-of-two method can smooth the jagged boundaries of foreground regions in those images very effectively.

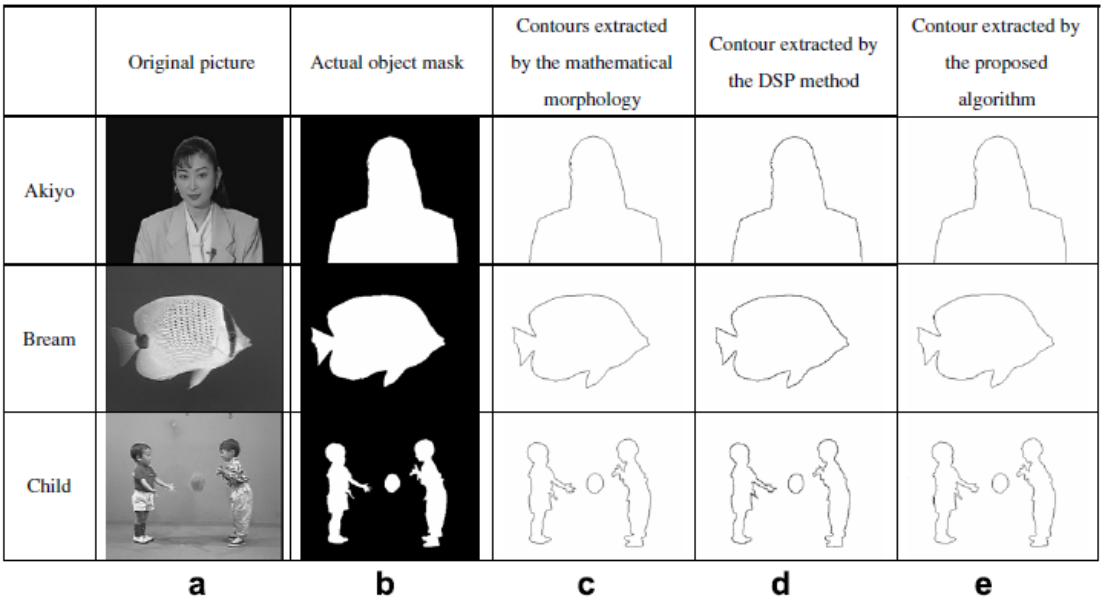


Fig. 7.83 – (a) Original ''Akiyo", ''Bream" and ''Children" images; (b) original contours; (c) contours extracted using mathematical morphology method; (d) contours extracted using the conventional DSP (Digital Signal Processing) method; and (e) contours extracted using PSCTM (Chan and Hsu, 2008).

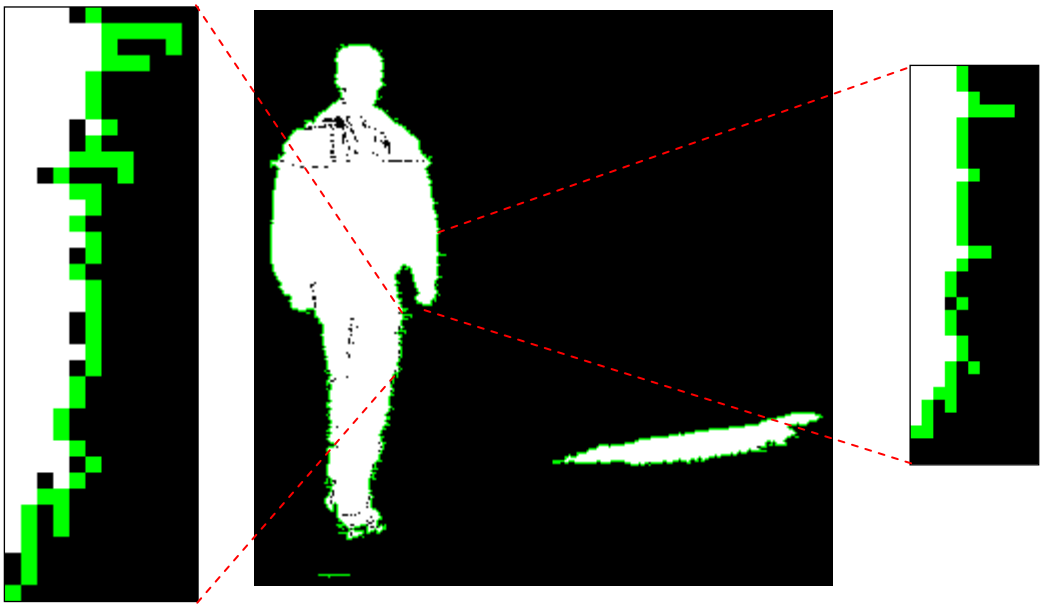
7.2.8 The results of applying the min-of-two method

In this section, the application of Quek's and the min-of-two algorithms on a number of foreground binary images is demonstrated. In addition, small parts of the output of the RLE smoothing program for Fld248 are given.

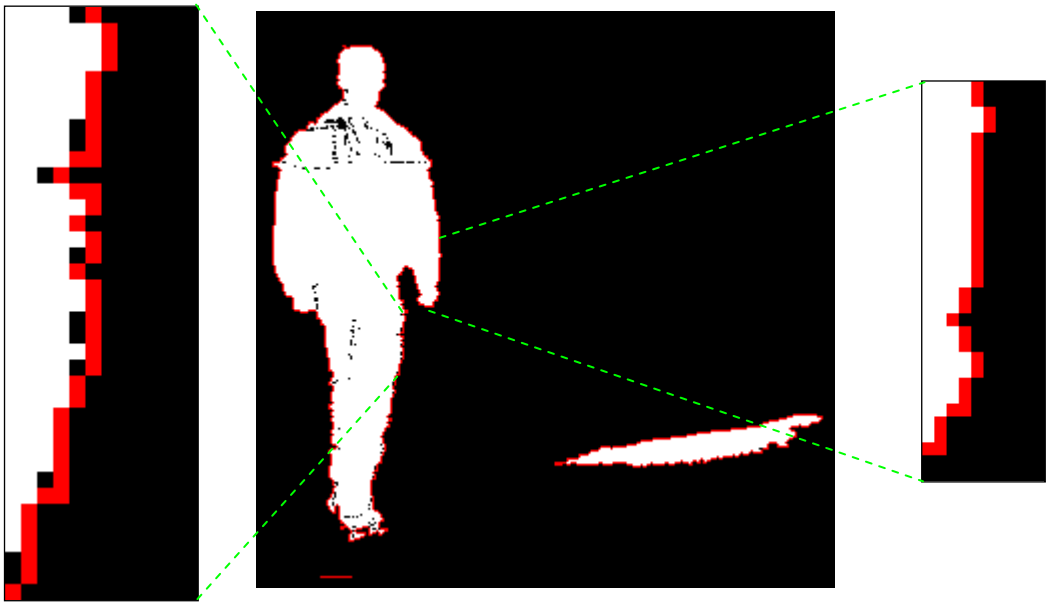
(a)

(b)

(c)



(d)



(e)

Fig. 7.84 – (a) Fld248 (b) Dif248 (c) Thresh248 (d) Sizefilter_Bound248 (e) Smoothed248; It is the result of applying the min-of-two smoothing method to Sizefilter_Bound248

The RLE smoothing program causes mass amount of output, so it takes a large number of pages to display all of them in this thesis. Thus, only very short parts of the output corresponding to Smoothed248 shown in Fig. 7.84 will be offered (the smoothed out-spike pixels are shown in grey colour):

####### 8-Connected Boundary Extraction and RLE Smoothing Program #######

File name: Thresh248.ppm
The no of regions remained after the size filter: 3

1. Region 11: Area=11195 First_run=17 Last_run=804
2. Region 227: Area=1488 First_run=609 Last_run=716
3. Region 244: Area=17 First_run=805 Last_run=807

The External Boundary of Region: 11

No. Trans Pcurr WALK Points Added (X, Y)

---

1. [2,3] (43, 17) RIGHT (44, 18)
2. [2,2] (44, 18) LEFT (43, 19)
3. [2,4] (43, 19) LEFT (42, 20) (41, 19)

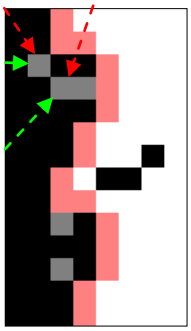
•
•
•

166. [2,2] (36,178) LEFT (36,179)

167. [2,2] (36,179) LEFT (35,180)
168. [2,3] (35,180) RIGHT (36,181)
169. [2,3] (36,181) RIGHT (37,181) (38,182)

Dn-P15: (5707)
bc[183]: dir=5 st_x=36 st_y=179 en_x=35 en_y=180 st_rn=552 en_rn=554
bc[184]: dir=7 st_x=35 st_y=180 en_x=36 en_y=181 st_rn=554 en_rn=558
bc[185]: dir=0 st_x=36 st_y=181 en_x=37 en_y=181 st_rn=558 en_rn=558
bc[186]: dir=7 st_x=37 st_y=181 en_x=38 en_y=182 st_rn=558 en_rn=561

Pixel (35, 180) removed.

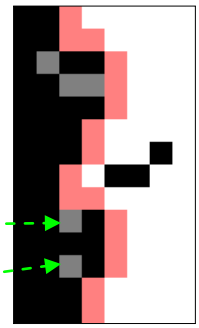


Repeating from step 167 by considering the removed pixel(s)
==========
167. [2,3] (36,179) RIGHT (37,179) (38,180)
168. [2,2] (38,180) LEFT (37,181) (36,181)
169. [2,3] (36,181) RIGHT (37,181) (38,182)

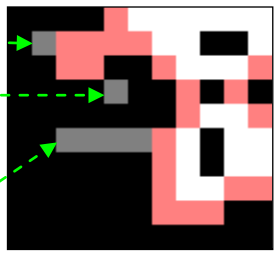
Dn-P1: (5407)
bc[185]: dir=5 st_x=38 st_y=180 en_x=37 en_y=181 st_rn=555 en_rn=558
bc[186]: dir=4 st_x=37 st_y=181 en_x=36 en_y=181 st_rn=558 en_rn=558
bc[187]: dir=0 st_x=36 st_y=181 en_x=37 en_y=181 st_rn=558 en_rn=558
bc[188]: dir=7 st_x=37 st_y=181 en_x=38 en_y=182 st_rn=558 en_rn=561

Pixels (36, 181) (37, 181) removed.

Repeating from step 168 by considering the removed pixel(s)
==========
168. [2,2] (38,180) LEFT (38,181)
169. [2,2] (38,181) LEFT (38,182)
170. [2,2] (38,182) LEFT (37,183)
171. [2,2] (37,183) LEFT (37,184)
172. [2,2] (37,184) LEFT (36,185)



•
Bm-P9:(5617)
Pixel (36, 187) removed.
•
Dn-P30:(537)
Pixel (36, 189) removed.
•
Dn-P4:(547)
Pixel (41, 257) removed.



•
Up-P20:(073)
Pixel (44, 259) removed.
•
Dn-P1:(5407)
Pixels (42, 261) (43, 261) (44, 261) (45, 261) removed.
•

Tp-P11:(0253)
Pixel (65, 257) removed.

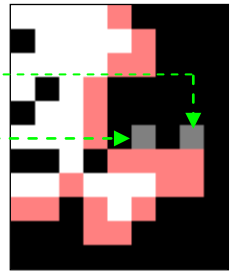
•

TP-P44:(435)
Pixel (63, 257) removed.

•

•

•



Bm-P26:(7621)
Pixels (71, 183) (71, 182) removed.

•

•

Up-P29:(13)
Pixel (73, 179) removed.

•

Up-P29:(13)
Pixel (72, 180) removed.

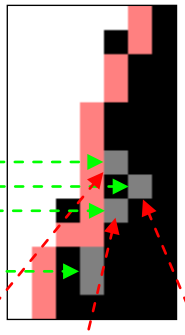
•

Up-P29:(13)
Pixel (72, 178) removed.

•

•

•



Another example of nested out-spikes

Bm-P6:(7243)
Pixel (75, 161) removed.

•

Up-P1:(1043)
Pixels (75, 160) (74, 160) removed.

•

Up-P29:(13)
Pixel (74, 158) removed.

•

Up-P1:(1043)
Pixels (76, 154) (75, 154) removed.

•

Bm-P6:(7243)
Pixel (78, 153) removed.

•

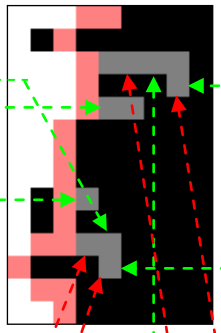
Up-P1:(1043)
Pixels (78, 152) (77, 152) (76, 152) (75, 152) removed.

•

•

•

•



Two examples of nested out-spikes

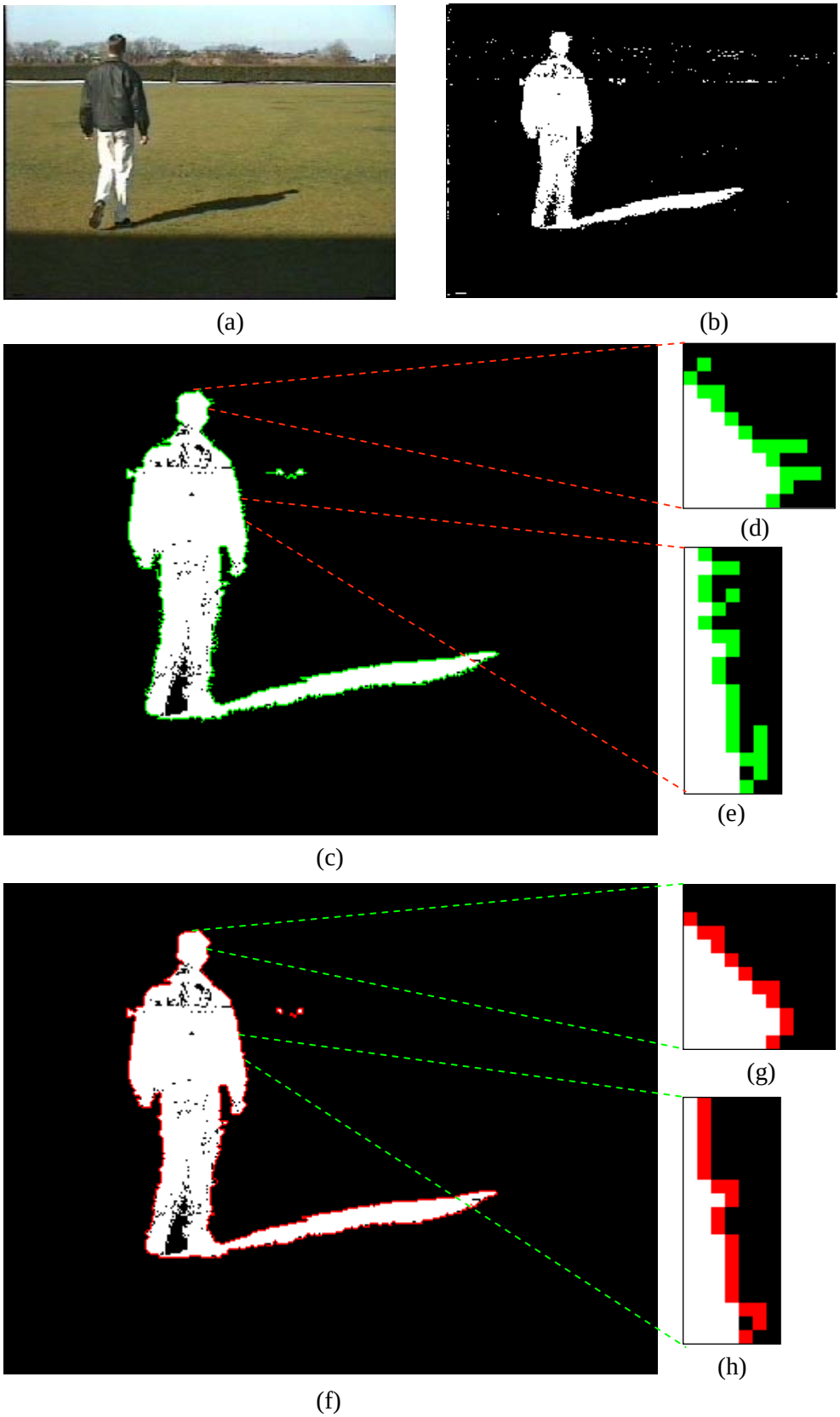


Fig. 7.85 – (a) Fld275 (b) Thresh275 (c) Sizefilter_Bound275 (d) A small segment of Sizefilter_Bound275 (e) Another small segment of Sizefilter_Bound275 (d) Smoothed275 (g) The same segment as (d) but from Smoothed275 (h) The same segment as (e) but from Smoothed275

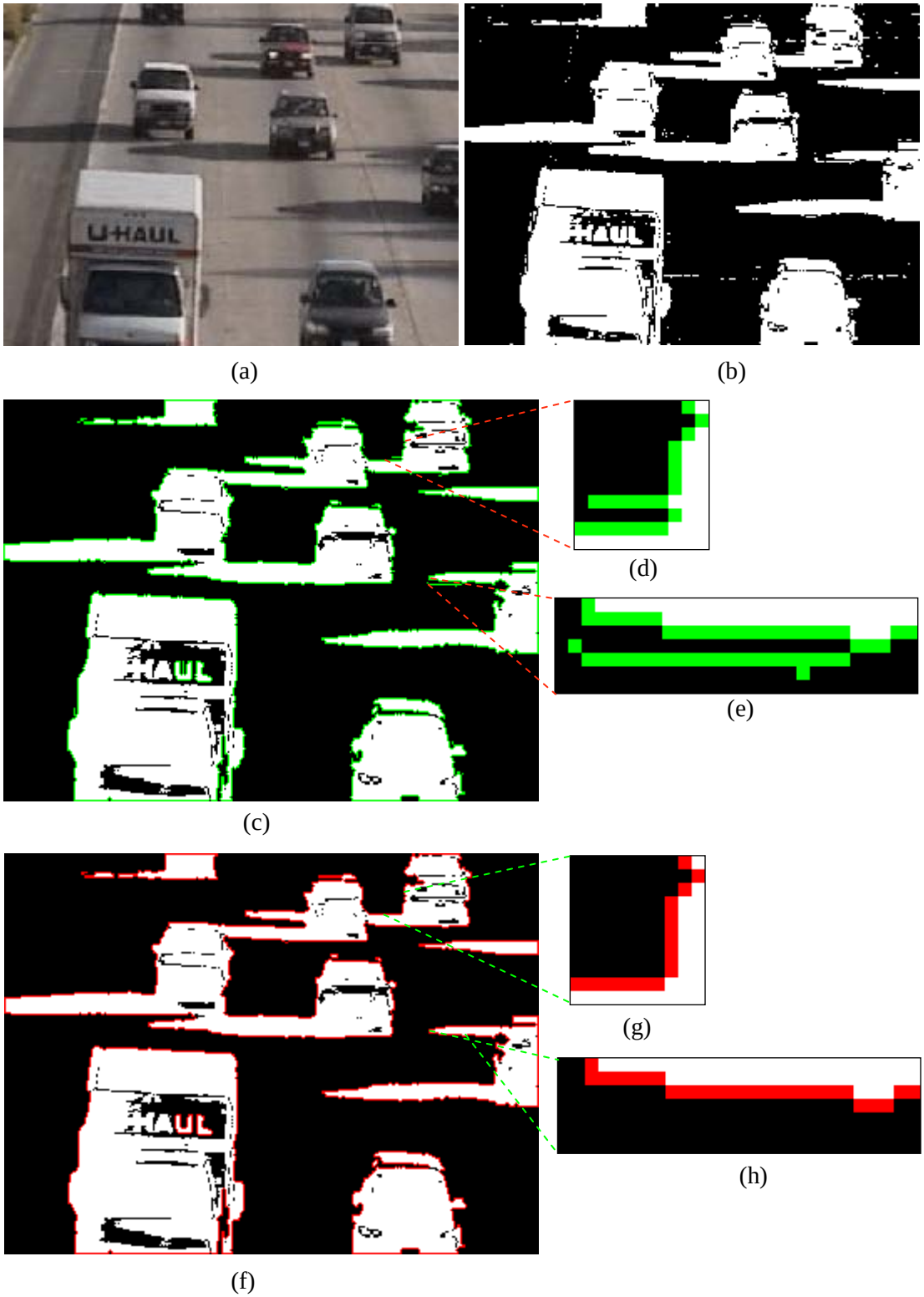


Fig. 7.86 – (a) Highway I-288 (b) Thresh288 (c) Sizefilter_Bound288 (d) A small segment of Sizefilter_Bound288 (e) Another small segment of Sizefilter_Bound288 (f) Smoothed288 (g) The same segment as (d) but from Smoothed288 (h) The same segment as (e) but from Smoothed288

The attempt has been on selecting only typical images containing large foregr-

ound objects (as large as possible) for this section. Thus, the effectiveness and efficiency of the min-of-two method will be easily observed in the smoothed images. For example, compare Fig. 7.84(d) with Fig. 7.84(e), Fig. 7.85(c) with Fig. 7.85(f) and Fig. 7.86(c) with Fig. 7.86(f). Just to mention, if a number of images consisting small foreground objects were selected, they may not only contain a few number of tiny out-spikes, but also the application of the min-of-two method to those out-spikes is not noticed easily. Although Quek's and the min-of-two algorithms can be applied to both external and internal boundaries in Fig. 7.85(c), they have been applied only to external boundaries. This is due to their more importance as some computations of foreground regions including the bounding box and the centre of gravity are obtained based on external boundaries.

7.2.9 The qualitative comparison of the min-of-two method with standard morphological operators

In this section, the "Closing" morphological operator is applied to the same thresholded images given in Section 7.2.8. Then the resulted outputs are qualitatively compared with the corresponding smoothed images resulted by the min-of-two method.

Based on the results from figures 7.87 to 7.89, it is concluded that the "Closing" morphological operator has the following disadvantages:

1. Once the "Closing" operator is applied to a thresholded binary image (i.e. after the size filter), noisy out-spikes are somewhat smoothed. However, the boundaries of foreground regions may not be smoothed completely. For example, figures 7.87(e), 7.88(f), 7.88(h), 7.89(e), and 7.89(f) show that there are still small out-spikes in the smoothed images. Compare these figures with Figs. 7.87(f), 7.88(g), 7.89(h), and 7.89(i) produced by the min-of-two method.

2. Figures 7.88(f) and 7.89(e) show that the "Closing" operator has distorted the boundaries of foreground regions so that the shapes of the resulted images are considerably eroded or even damaged in comparison with their corresponding thresholded images (i.e. in comparison with 7.88(b) and 7.89(b), respectively. Also compare those figures with 7.88(g) and 7.89(h)).

3. The erosion of a number of foreground pixels may in some cases cause a number of foreground regions to split in two or more regions. For example, Fig. 7.89(f) shows such an issue. As a result, this effect produces severe difficulties for later processing such as tracking and recognition phases. However, there is no split in its corresponding figure 7.89(i).

4. In addition, the "Closing" operator has eliminated a large number of pixels from the internal parts of foreground regions. The segments shown inside ovals in figures 7.87(c) and 7.88(d) are examples of this effect. Compare these images with corresponding thresholded images in Figs. 7.87(a) and 7.88(a), respectively. Meanwhile, the segments inside ovals in figures 7.87(d) and 7.88(e) are the same as the internal parts of ovals in figures 7.87(a) and 7.88(a).

Thus, based on the above issues, the min-of-two smoothing method has none of the above difficulties. It just removes all boundary out-spikes. It does not produce any undesired changes and has no effect on the other boundary pixels or the internal parts of the foreground regions. Besides, it is almost imposible for the min-of-two method to split foreground regions. It produces smoothed regions which have very similar shapes in comparison with their original regions in the thresholded images. Meahwhile, other morphological operators including “Opening” or “two erosions followed by one dilation” produce worse results than the “Closing” operator. Therefore, this proves the superiority of the min-of-two method over the standard morphological operators.

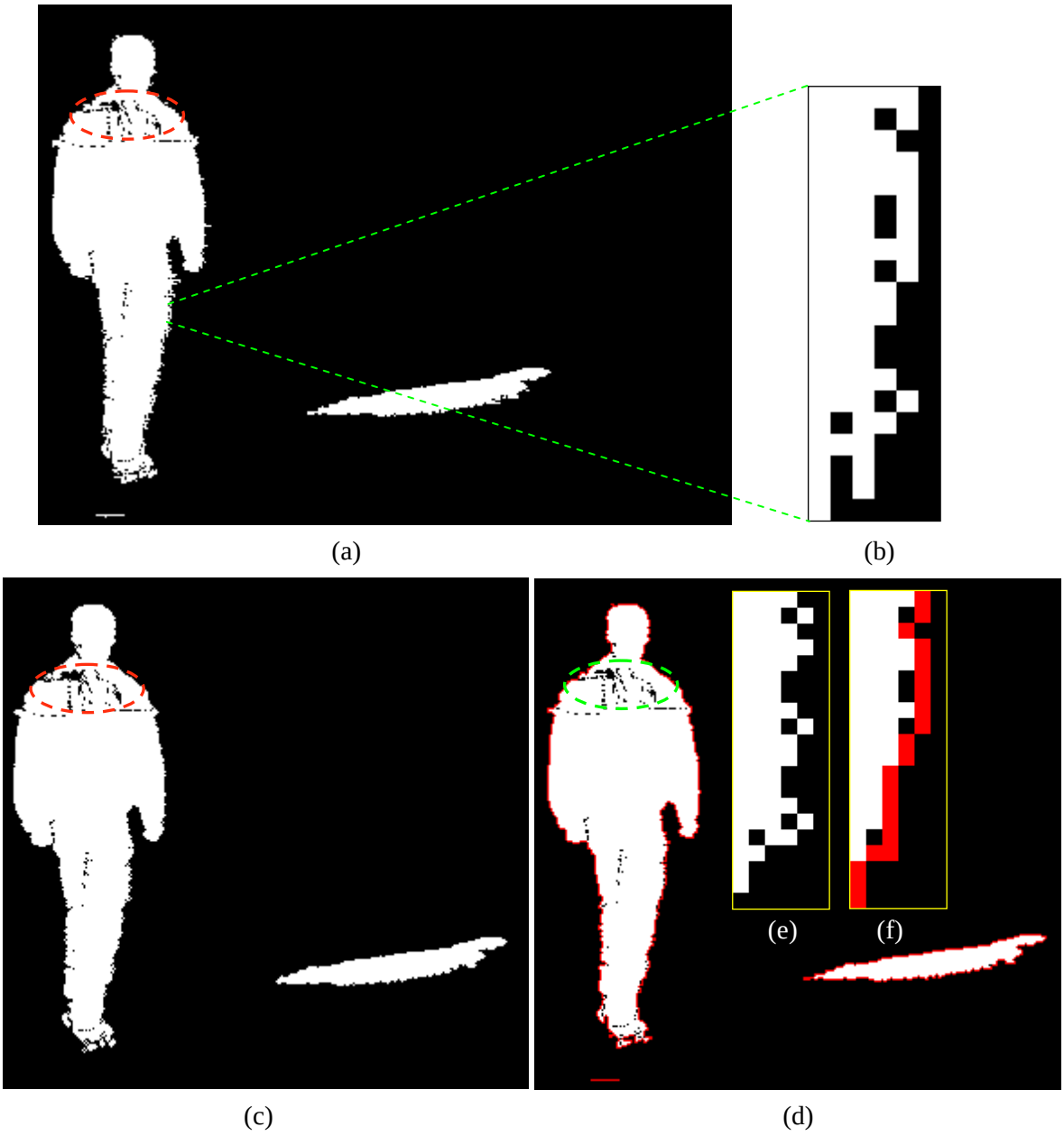


Fig. 7.87 – (a) Sizefilter248 (b) A small segment of Sizefilter248
(c) Closing248 (d) Smoothed248 (e) The same small segment as (b) but from Closing248
(f) The same small segment as (b) but from Smoothedg248

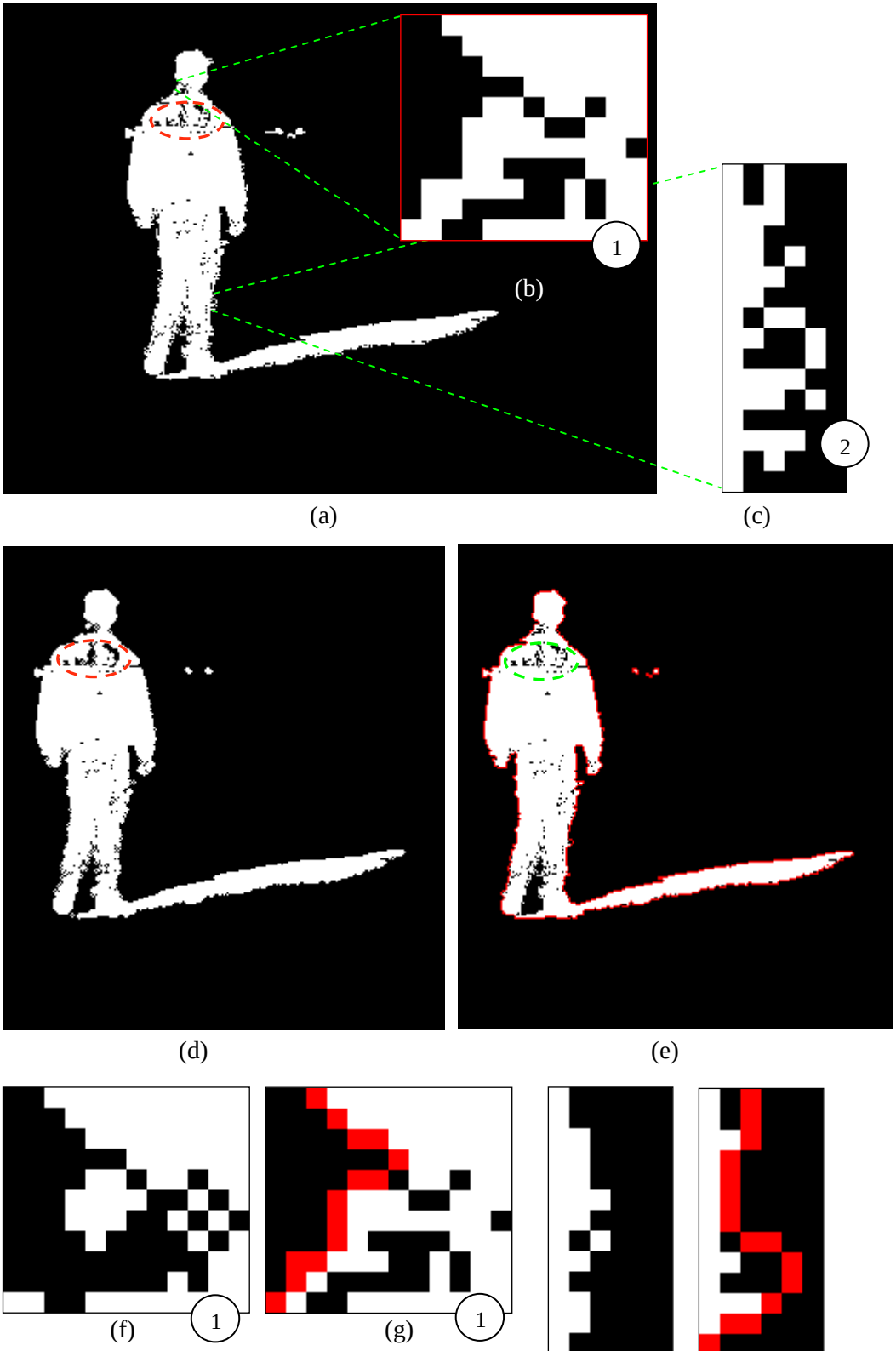


Fig. 7.88 – (a) Sizefilter275 (b) A small segment of Sizefilter275 (c) Another small segment of Sizefilter275 (d) Closing275 (e) Smoothed275 (f) The same segment as (b) but from (d) (g) The same segment as (b) but from (e) (h) The same segment as (c) but from (d) (i) The same segment as (c) but from (e)

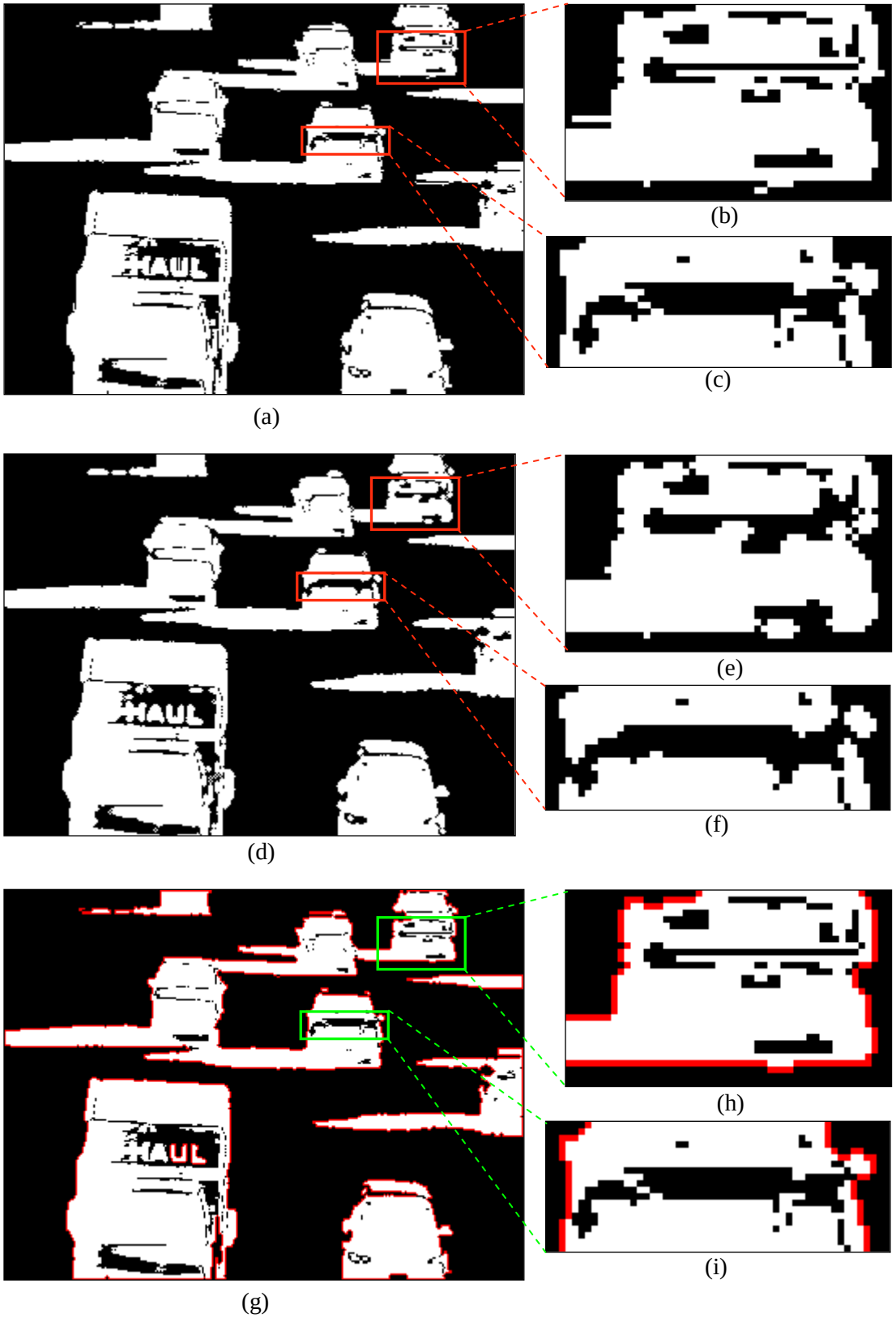


Fig. 7.89 – (a) Highway I-Sizefilter288 (b) A segment of Highway I-Sizefilter288 (c) Another segment of Highway I-Sizefilter288 (d) Highway I-Closing288 (e) The same segment as (b) but from (d) (f) The same segment as (c) but from (d) (g) Highway I-Smoothed288 (h) The same segment as (b) but from (g) (i) The same segment as (c) but from (g)

7.2.10 The quantitative comparison of the min-of-two method with a standard morphological operator

The min-of-two method was compared with chain code smoothing algorithms operating on binary images qualitatively in Section 7.2.7.1. The result was that each algorithm was in some way different or incomparable with the min-of-two RLE smoothing method. As a result, a quantitative comparison can be performed only between the min-of-two method and a standard morphological operator such as the Closing operator as given in this section. The qualitative comparison results between the Closing operator and the min-of-two method were offered in Section 7.2.9.

For a quantitative comparison, more than 100 thresholded frames after the size filter of *Fld*, *Highway I*, and *Bijan1* image sequences were selected. Then, Quek and the min-of-two methods and the Closing operator were applied to them separately and their total processing times were computed as shown in Table 7.2. Compuatation times were obtained using a 2.4 GHz Pentium 4 PC with 1 GB of RAM running windows XP professional edition.

Because some parts of the min-of-two method were completed while Quek algorithm was operating, thus, the times in Table 7.2 were computed for their combination and not for the min-of-two method alone.

Based on the information given in Table 7.2, the average time of RLE boundary tracing and smoothing of each frame is less 0.4 ms. On the other hand, the average time of smoothing each frame using the Closing operator is at least about 4.9 ms or more. This shows that Quek and the min-of-two smoothing mthod can not only run very rapidly but also are at least 12 times faster than the Closing operator.

Therefore, based on the results of Section 7.2.9 and Table 7.2, the superiority and high effectiveness of Quek and the min-of-two methods in comparison to the Closing operator from both qualitative and quantitative point of view are confirmed.

		Quek and the Min-of-two Methods		**The Closing Operator**	
Image Sequence	**No of Frames**	**Total Time (ms)**	**Average Time (ms)**	**Total Time (ms)**	**Average Time (ms)**
Fld	110	41.2	0.37	750.0	6.82
Highway I	110	32.9	0.30	545.0	4.95
Bijan1	110	27.5	0.25	717.0	6.52

Table 7.2 – Computation times of Quek and the min-of-two methods and the Closing operator for three image sequences *Fld*, *Highway I* and *Bijan1*.

Chapter 8

Conclusions and Future Work

The main goal of this thesis is to review and offer robust and efficient algorithms for the detection (or the segmentation) of foreground regions (or objects) in indoor and outdoor scenes using colour image sequences captured with a stationary camera. For this purpose, the block diagram of a simple vision system is offered in Chapter 2. There are two goals for this block diagram (Fig. 2.1). Firstly, it gives the idea of the precise order of blocks and their tasks, which should be performed to detect moving foreground objects. Secondly, a check mark (✓) on the top right corner of a block indicates a relevant research about it.

A number of chapters of this thesis are devoted to dynamic background generation for colour video sequences. Often a large number of pixels of each input image are background pixels. In fact, recognition and separation background areas have a key role in the segmentation of foreground regions. In this work, a review of the effectiveness of a number of important background algorithms, along with their major features, are presented. In addition, two background approaches are offered. The first approach is a pixel-based technique whereas the second one works at object level. For each approach, three algorithms are presented. They are called the 'Selective Update Using Non-Foreground Pixels of the Input Image', the 'Selective Update Using Temporal Averaging' and the 'Selective Update Using Temporal Median', respectively in this thesis. The first approach has some deficiencies, which makes it incapable to produce correct dynamic background images. Three methods of the second approach use an invariant colour filter and a suitable motion tracking technique, which selectively exclude foreground objects (or blobs) from the background frames. The difference between the three algorithms of the second approach is in updating method of the background pixels.

It is shown that the third method produces the correct background image for each input frame. The advantages of this method are as it operates in unconstrained outdoor and indoor scenes. Also it is able to handle difficult situations such as removing ghosts and including stationary objects in the background image effectively. Meanwhile, the algorithm's parameters are computed automatically or

are fixed. Very good results obtained on a number of image sequences confirm the effectivity of the new algorithm.

After background subtraction, the colour difference image is binarised by thresholding. The accurate selection of a thresholding technique as a suitable image segmentation tool has a major effect on correctly identifying moving foreground objects. For this purpose, a discussion about an optimum thresholding method for colour difference image is given in Chapter 4. Then, in the binary image, 1-pixels are grouped together by a connected components labelling operator in order to identify the connected foreground regions. Due to emphasis on simplicity in this work, a sequential method based on run-length encoding (RLE) has been used (Haralick and Shapiro, 1992, Chapter 2).

Representing foreground regions using their boundaries is also an important task. Thus, a contour tracing algorithm using RLE data (Quek, 2000) is implemented successfully to obtain the outer and inner boundaries of the objects. The main reason for using an RLE contour tracer is that an RLE format already used for representing internal characteristics of foreground objects. After the thresholding process, the boundaries of foreground regions often have jagged appearances. Thus, foreground regions may not correctly be recognised reliably due to their corrupted boundaries. A very efficient boundary smoothing method based on RLE data is proposed in Chapter 7. It just smoothes the external and internal boundaries of foreground objects and does not distort the silhouettes of foreground objects. As a result, it is very fast and does not blur the image.

As the future work, several achievements may be performed as follows:

1. The proposed background model requires to be extended so that it can also handle multi-modal background models.

2. Based on the strengths and weaknesses of a number of shadow detection algorithms given in Chapter 2 (especially the recent ones), a suitable method should be designed to match with the other blocks of Fig. 2.1.

3. A tracking method based on the algorithm of Fuentes and Velastin (2001) was already implemented and applied by the proposed background subtraction method in this thesis. However, an investigation should be performed to check whether it can handle difficult situations such as occlusion or disocclusion of foreground objects. If it is not sufficiently capable to manage difficult situations, a more robust and effective technique should be sought or proposed.

4. After silhouette smoothing (block no. 10 in Fig. 2.1), the required information about foreground objects is available. Then, based on a predefined scenario for a specific application and the foreground objects' information (as suggested in section 2.2.11), a high-level methodology should be designed so that the activities and behaviours of foreground objects are understood and interpreted.

Appendix A

Summaries of Two Background Subtraction Algorithms

In this apendix, summaries of two background subtraction algorithms are given. This is because those two algorithms are compared with the proposed background method in Chapter 6.

A.1 Horprasert et al.'s algorithm

Horprasert et al. (1999) presents a pixel-based colour model which separates the brightness component and the chromaticity component. For each pixel i, E_i = [$E_R(i)$, $E_G(i)$, $E_B(i)$] represents the pixel's expected *RGB* colour in the background image and I_i = [$I_R(i)$, $I_G(i)$, $I_B(i)$] denotes the pixel's *RGB* colour value in the input image. The distortion of I_i is measured with respect to E_i. For a pixel i, this distortion is decomposed into two components *brightness distortion* (α_i) and *chromaticity distortion* (CD_i) which are given by the following equations:

$$\alpha_i = \frac{\left(\frac{I_R(i)\mu_R(i)}{\sigma_R^2(i)} + \frac{I_G(i)\mu_G(i)}{\sigma_G^2(i)} + \frac{I_B(i)\mu_B(i)}{\sigma_B^2(i)} \right)}{\left(\left[\frac{\mu_R(i)}{\sigma_R(i)} \right]^2 + \left[\frac{\mu_G(i)}{\sigma_G(i)} \right]^2 + \left[\frac{\mu_B(i)}{\sigma_B(i)} \right]^2 \right)} \qquad \text{(Eq. A.1)}$$

$$CD_i = \sqrt{\left(\frac{I_R(i) - \alpha_i\mu_R(i)}{\sigma_R(i)} \right)^2 + \left(\frac{I_G(i) - \alpha_i\mu_G(i)}{\sigma_G(i)} \right)^2 + \left(\frac{I_B(i) - \alpha_i\mu_B(i)}{\sigma_B(i)} \right)^2} \qquad \text{(Eq. A.2)}$$

where $\mu_R(i)$, $\mu_G(i)$, and $\mu_B(i)$ are the arithmetic means and $\sigma_R(i)$, $\sigma_G(i)$, and $\sigma_B(i)$ are the standard deviations of the ith pixel's red, green, blue values computed over N background frames. In the background training process, the reference background image and some parameters associated with normalisation are computed over a number of *static background* frames. A pixel is modelled by a 4-tuple < E_i, s_i, a_i, b_i>

where E_i is the expected color value, s_i is the standard deviation of colour value, a_i is the variation of the brightness distortion, and b_i is the variation of the chromaticity distortion of the ith pixel, which are given by the following equations:

$$a_i = RMS(\alpha_{ji}) = \sqrt{\frac{\sum_{j=0}^{N}(\alpha_{ji} - 1)^2}{N}}, \quad j = \text{frame number}, 0 \le j \le N \qquad \text{(Eq. A.3)}$$

$$b_i = RMS(CD_{ji}) = \sqrt{\frac{\sum_{j=0}^{N}(CD_{ji})^2}{N}} \qquad \text{(Eq. A.4)}$$

Since different pixels yield different distributions of α_i and CD_i, in order to use a single threshold for all the pixels, α_i and CD_i are rescaled as follows:

$$\hat{\alpha}_{ji} = \frac{\alpha_{ji} - 1}{a_i}, \qquad j = \text{frame number}, \quad 0 \le j \le N \qquad \text{(Eq. A.5)}$$

$$\hat{CD}_{ji} = \frac{CD_{ji}}{b_i} \qquad \text{(Eq. A.6)}$$

where $\hat{\alpha}_{ji}$ and $\hat{CD}_{ji}$ are called *normalised brightness distortion* and *normalised chromaticity distortion*, respectively.

There might be a case where a pixel from a moving object in current image contains very low *RGB* values. This dark pixel will always be misclassified as a shadow. To avoid this problem, a lower bound ($\tau_{\alpha lo}$) is introduced for the normalised brightness distortion. Based on the above definitions, a pixel is classified into one of the four categories {*B, F, S, H*} (i.e. background, foreground, shadow, and highlight) by the following decision procedure:

$$M(i) = \begin{cases} F: \ \hat{CD}_{ji} > \tau_{CD} \text{ or } \hat{\alpha}_{ji} < \tau_{\alpha lo}, \ else \\ B: \ \hat{\alpha}_{ji} < \tau_{\alpha 1} \ and \ \hat{\alpha}_{ji} > \tau_{\alpha 2}, \ else \\ S: \ \hat{\alpha}_{ji} < 0, \ else \\ H: \end{cases}, \text{ for } j \ge N + 1 \text{ as above} \qquad \text{(Eq. A.7)}$$

References

Agrawal A., Nekludova L., and Lim W., 1987. “A parallel O(logN) algorithm for finding connected components in planar images”, in Proceedings of International Conference on Parallel Processing, pp. 783-786.

Agrawala A. K. and Kulkarni A. V., 1977, “A sequential approach to the extraction of shape features”, Computer Graphics and Image Processing, vol. 6, pp. 538-557.

Anderson C., Burt P., and van der Wal G., 1985, “Change detection and tracking using pyramid transformation techniques”, In Proc. SPIE Intelligent Robots and Computer Vision, vol. 579, pp. 72–78.

Bejanin M., Huertas A., Medioni G., and Nevatia R., 1994, “Model validation for change detection," In Proceedings 2nd Int. IEEE Workshop on Applications of Computer Vision, pp. 160-167.

Bernsen J., 1986. “Dynamic Thresholding of Grey-level Images”, In Proceedings 8th International Conference on Pattern Recognition, pp. 1251-1255.

Boult T. and et al., 1999, “Frame-rate omnidirectional surveillance and tracking of camuaged and occluded targets”, In Proceedings second IEEE workshop on Visual Surveillance, pp. 48-55.

Brown L. G., 1992. “A survey of image registration techniques”, ACM Computer Survey, vol. 24, no. 4, pp. 325-376.

Cai Z., 1988, “Restoration of binary images using contour direction chain codes description”, Computer Vision Graphics and Image Processing, vol. 41, pp. 101–106.

Capson D. W., 1984, “An improved algorithm for the sequential extraction of boundaries from a raster scan”, Computer Vision, Graphics Image Process, vol. 28, pp. 109-125.

Carmona E., Martinez-Cantos J., and Mira J., 2008, “A new video segmentation method of moving objects based on blob-level knowledge”, Pattern Recognition Letters, vol. 29, pp. 272–285.

Cavallaro A., Salvador E., and Ebrahimi T., 2005, “Shadow-aware object-based video processing”, IEE Vision, Image and Signal Processing, Vol. 152, no. 4, pp. 14- 22.

Chan D-Y. and Hsu R. C., 2008, “Robust shape-preserving contour tracing with synchronous redundancy pruning”, Pattern Recognition Letters, vol. 29, no. 5, pp. 569-579.

Chang L. W. and Leu K. L., 1990, “A fast algorithm for restoration of images based on chain codes description and its applications”, Computer Vision Graphics and Image Processing, vol. 50, pp. 296-307.

Chen K, 1999, “A feature preserving adaptive smoothing method for early vision”, Technical Report, Peking University, China.

Cheng Y., Jensen J. R., Huntsberger T., and Huntsberger B., 1994. “Hypercube algorithm for image component labeling,” in Proceedings of the Scalable High-Performance Computing Conference, pp. 259-262.

Cheng J, Yang J., Zhou Y., and Cui Y., 2006, “Flexible background mixture models for foreground segmentation”, Image and Vision Computing, vol. 24, no. xx, pp. 473-482.

Cheung S. -C. and Kamath C., 2004, “Robust techniques for background subtraction in urban traffic video”, In Proceedings of SPIE (5308): Visual Communications and Image Processing, pp. 881-892.

Chia T. -L, Wang K. –B., Chen L. –R., and Chen Z., 2003, “A parallel algorithm for generating chain code of objects in binary images”, Information Sciences, vol. 149, no. 4, pp. 219-234.

Choi J., Yoo Y. J., and Choi j. Y., 2010, “Adaptive shadow estimator for removing shadow of moving object”, Computer Vision and Image Understanding, vol. 114, pp. 1017-1029.

Chow C. K. and Kaneko T, 1972. “Automatic Detection of the Left Ventricle from Cineangiograms”, in Computers and Biomedical Research, vol. 5, pp. 388-410.

Collins R., Lipton A., Kanade T., Fujiyoshi H., Duggins D., Tsin Y., Tolliver D., Enomoto N., and Hasegawa O., 2000, “A system for video surveillance and monitoring: VSAM final report”, Robotics Inst., Technical Report, CMU-RI-TR-00-12, Carnegie Mellon University, USA.

Comaniciu D. and Meer P., 2002, “Mean shift: a robust approach toward feature space analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. 603–619.

Cristani M., Bicego M., and Murino V., 2003, “Multi-level background initialization using Hidden Markov Models”, In First ACM SIGMM Int. workshop on Video surveillance, pp. 11-20.

Cucchiara, R. Grana, C., Piccardi, M., and Prati A., 2000, “Statistic and knowledge-based moving object detection in traffic scenes”, In Proc. of Intelligent Transportation Systems Conference, pp. 27-32.

Cucchiara, R. Grana, C., Piccardi, M., and Prati A., 2001, “Detecting objects, shadows and ghosts in video streams by exploiting color and motion information”, In Proc. of 11th International Conference on Image Analysis and Processing, pp. 360-365.

Cucchiara R., Grana C., Piccardi M., and Prati A., 2003, “Detecting moving objects, ghosts and shadows in video streams”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 10, pp. 1337-1342.

Dagless E. L., Ali A. T., and Bulas Cruz, J., 1993, “Visual road traffic monitoring and data collection”, In Proc. of IEEE-IEE Vehicle Navigation and Information Systems, pp. 146-149.

Dawson-Howe K., 1996, “Active Surveillance using Dynamic Background Subtraction”, Technical Report No. TCD-CS-96-06, Trinity College of Dublin, Ireland.

Di Zenzo S., Cinque L., and Levialdi S., 1996, “Run-based algorithms for binary image analysis and processing”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, no. 1, pp. 83-89.

Dorst L. and Smeulders A. W. M., 1987, “Length estimated for digitized contours”, Computer Graphics and Image Processing, vol. 40, pp. 317-333.

Du Y., Chang C. I., and Thouin P. D., 2004, “Unsupervised approach to color video thresholding”, Optical Engineering, vol. 43, no. 2, pp. 282-289.

Eikvil L., Taxt T., and Moen K, 1991. “A Fast Adaptive Method for Binarization of Document Images”, in Proceedings the 1st International Conference on Document Analysis and Recognition, pp. 453-443.

Elgammal, A., Duraiswami R. ,Harwood, D., and Davis, L.S., 2002, “Background and foreground modeling using nonparametric kernel density estimation for visual surveillance”, Proceedings of the IEEE, vol. 90, no. 7, pp. 1151–1163.

Elgammal, A., Harwood, D., and Davis, L.S., 2000, “Non-parametric Model for Background Subtraction", In Proc. of European Conference on Computer Vision, vol. 2, 751-767.

Elhabian S., El-Sayed K. and Ahmed S., 2008, “Moving Object Detection in Spatial Domain using Background Removal Techniques - State-of-Art”, Recent Patents on Computer Science, vol. 1, no. 1, pp. 32-54.

Eng H., Wang J., Kam A., and Yau W., 2004, “Novel region-based modeling for Human Detection within High Dynamic Aquatic Environment” In Proc. on Computer Vision and Pattern Recognition, vol. 2, pp. 390-397.

Fang L., Qiong W., and Sheng Y., 2008, “A method to segment moving vehicle cast shadow based on wavelet transform”, Pattern Recognition Letters, vol. 29, pp. 2182–2188.

Fejes S. and Davis L. S., 1998, “What can projections of flow fields tell us about the visual motion”, in Proceedings of International Conference on Computer Vision, pp. 979-986.

Figueiredo M. and Jain A., 2002, “Unsupervised learning of finite mixture Models”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 3, pp. 381-396.

Francis K. H. Q., 1990, “On Three-dimensional object recognition and pose determination: an abstraction based approach”, Ph.D. Thesis, the University of Michigan, MI.

Freeman H., 1961, “On the encoding of arbitrary geometric configurations”, IRE Transactions on Electron. and Comput., vol. 10, pp. 260-268.

Fuentes L. M. and Velastin S. A., 2006, “People tracking in surveillance applications”, Image and Vision Computing, vol. 24, no. 11, pp. 1165-1171.

Garnica C., Boochs F., Twardochlib M., 2000. “A New Approach to Edge-preserving Smoothing for Edge Extraction and Image Segmentation”, in Proceedings of International Archives of Photogrammetry and Remote Sensing, vol. XXXIII, page 1.

Gevers T. and Smeulders A. W. M., 1999, “Colour-based object recognition”, Pattern Recognition, vol. 32, pp. 453–464.

Gonzalez R. C. and Wintz P., 1987, “Digital Image Processing”, Addison-Wesley Reading, MA.

Grant G. and Reid A. F., 1981, “An efficient algorithm for the boundary tracing and feature extraction”, Computer Graphics and Image Processing, vol. 17, pp. 225-237.

Grimson E., Stauffer C., Roman R., and Lee L., 1998, “Using adaptive tracking to classify and monitoring activities in a site”, in IEEE Conf. on Computer Vision and Pattern Recognition, pp. 22–29.

Gutchess D., Trajkovic M., Cohen-Sola E., Lyons D., and Jain A. K., 2001, “A background model initialization algorithm for video surveillance," in Proceedings Eighth IEEE International Conference on Computer Vision, vol. 1, pp. 744-750.

Halevy G. and Weinshall D., 1999, “Motion of disturbances: detection and tracking of multi-body non-rigid motion”, Machine Vision and Applications, vol. 11, pp. 122-137.

Han B., Comaniciu D., and Davis L., 2004, “Sequential kernel density approximation through mode propagation: Applications to background modeling”, In Proc. of Asian Conference on Computer Vision.

Harville M., 2002, “A framework for high-level feedback to adaptive, per-pixel, mixture-of-Gaussian background models”, In Proceedings of the Seventh European Conference on Computer Vision, Part III, pp. 543-560.

Haralick R. M., Shapiro L. G., 1992. Computer and Robot Vision, Volume I, Addison-Wesley Publishing Company.

Haritaoglu I., Harwood D. and Davis L. S., 2000, “W^4: Real-time surveillance of people and their activities”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 8, pp. 809-830.

Hildreth E. C., 1983, “The detection of intensity changes by computer and biological vision systems”, Computer Vision, Graphics and Image Processing, vol. 22, pp. 1–27.

Hong, D. and Woo, W., 2003, “A background subtraction for a vision-based user interface”, In Proceedings of ICICS-PCM, Singapore, pp. 1-5.

Horprasert T., Harwood D., and Davis L. S., 1999, “A statistical approach for real-time robust background subtraction and shadow detection,” In Proc. IEEE Frame-Rate ApplicationsWorkshop, Greece.

Hou Z. and Han C., 2004, “A background reconstruction algorithm based on pixel intensity classification in remote video surveillance system”, in Proceedings of the 7th International Conference on Information Fusion, Sweden, pp. 754-759.

Hu J., Yu D., and Yan H., 1998, “A multiple point boundary smoothing algorithm”, Pattern Recognition Letters, vol. 19, no. 8, pp. 657-668.

Huang J.-B. and Chen C.-S., 2009, “Moving Cast Shadow Detection using Physics-based Features”, IEEE Conf. on Computer Vision and Pattern Recognition, pp. 2310-2317.

Huang L. and Wang M., 1995, “Image thresholding by minimizing the measures of fuzziness’’, Pattern Recognition, vol. 28, pp. 41–51.

Iannizzotto G. and Vita L., 2000, “Fast and accurate edge-based segmentation with no contour smoothing in 2-D real images”, IEEE Transactions on Image processing, vol. 9, no. 7, pp. 1232-1237.

Ibanez L., Schroeder W., Ng L., and Cates J., 2003, “The ITK Software Guide: The Insight Segmentation and Registration Toolkit (version 1.4)”, Kitware Inc.

Irvin R. and Mckeown D., 1989, “Methods for exploiting the relationship between buildings and their shadows in aerial imagery”, IEEE Transactions on Systems, Man, Cybernetics, vol. 19, no. 6, pp. 1564-1575.

Javed O., Shafique K., and Shah M., 2002, “A hierarchical approach to robust background subtraction using color and gradient information”, In Proceedings of IEEE Workshop on Motion and Video Computing, pp. 22-27.

Jain R., 1981, “Extraction of motion information from peripheral processes”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 3, pp. 489-503.

Jain R., Kasturi R., and Shunck B., 1995, “Machine Vision”, McGraw-Hill Inc., New York.

Jiang C. and Ward M. O., 1994, “Shadow Segmentation and Classification in a Constrained Environment”, CVGIP: Image Understanding, vol. 59, no. 2, pp. 213-225.

Johansson B., Wiklund J., Forssen P., and Granlund G., 2009, “Combining shadow detection and simulation for estimation of vehicle size and position”, Pattern Recognition Letters, vol. 30, pp. 751–759.

Jolly M. -P. D., Lakshmanan S. and Jain A. K., 1996, “Vehicle segmentation and classification using deformable templates”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, no. 3, pp. 293-308.

Joshi A. and Papanikolopoulos N., 2008, “Learning to Detect Moving Shadows in Dynamic Environments”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 11, pp. 2055-2063.

Jung C., 2009, “Efficient Background Subtraction and Shadow Removal for Monochromatic Video Sequences”, IEEE Transactions on Multimedia, vol. 11, no. 3, pp. 571-577.

Kaewtrakulpong P., and Bowden R., 2001, “An improved adaptive background mixture model for real-time tracking with shadow detection,” In Proceedings of 2nd European Workshop on Advanced Video Based Surveillance Systems.

Kampel M., H. Wildenauer H., Blauensteiner P., and Hanbury A., 2007, “Improved motion segmentation based on shadow detection”, Electronic Letters on Computer Vision and Image Analysis, vol. 6, no. 3, pp. 1-12.

Kapur J., Sahoo P., and Wong A., 1985, “A new method for gray-level picture thresholding using the entropy of the histogram”, Graphical Models and Image Processing, vol. 29, pp. 273–285.

Karmann K. -P. and von Brandt A., 1990, “Moving object recognition using and adaptive background memory”, In Time-Varying Image Processing and Moving Object Recognition, Cappellini V., 2nd ed., pp. 289-307.

Kim K., Chalidabhongse T., Harwood D., and Davis L., 2005, “Real-time foreground-background segmentation using codebook model”, Real Time Imaging, vol. 11, no. 3, pp. 172–185.

Kim M., Jeon J. G., Kwak J. S., Lee M. H., and Ahn C., 2001, “Moving object segmentation in video sequences by user interaction and automatic object tracking”, Image and Vision Computing, vol. 19, no.5, pp. 245-260.

Kim S. -D., Lee J. -H., and Kim J. -K., 1988, “A new chain-coding algorithm for binary images using run-lengthy codes”, Computer Vision, Graphics, Image Processing, vol. 41, pp. 114-128.

Kittler J. and Illingworth J., 1986, “Minimum error thresholding”, Pattern Recognition, vol. 19, no.1, pp. 41–47.

Klaus B. and Horn P., 1986, “Robot Vision”, MIT Press, Cambridge, MA.

Koller D., Danilidis K., and Nagel H.-H., 1993, “Model-based object tracking in monocular image sequences of road traffic scenes”, International Journal of Computer Vision, vol. 10, no. 3, pp. 257-281.

Koller D., Weber J., Huang T., Malik J., Ogasawara G., Rao B., and Russell S., 1994, “Towards Robust Automatic Traffic Scene Analysis in Real-time”, In Proc. ICPR’94, pp. 126-131.

Kompatsiaris I., and Strintzis M. G., 2000, “Spatiotemporal segmentation and tracking of objects for visualization of videoconference image sequences”, IEEE Transactions on Circuit and Systems for Video Technology, vol. 10, no. 8, pp. 1388-1403.

Koplpwitz J. and Deleone J., 1996, “Hierarchical representation of chain-encoded binary image contours”, Computer Vision and Image Understanding vol. 63, no. 2, pp. 344–352.

Lambert S., de Leau E., and Vuurpijl L., 1999, “Using pen-based outlines for object-based annotation and image-based queries”, In Proceedings of the Third International Conference on Visual Information and Information Systems, pp. 585-592.

Lavallee S., 1995, “Registration for computer-integrated surgery: methodology, state of the art”, MIT Press.

Legault R. and Suen C. Y., 1997, “optimal local weighted averaging methods in contour following”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 8, pp. 801-817.

Levner I., 2002, “Shape detection, analysis and recognition”, Technical Report, TR02-18, University of Alberta, Canada.

Li H. F., Jayakumar R., and Youssef R., 1989, “Parallel algorithms for recognizing handwritten characters using shape features”, Pattern Recognition, vol. 22, no. 6, pp. 641-652.

Lindley C. A., 1991, “Practical Image Processing in C”, Wiley, New York.

Liow Y. -T., 1991, “A contour tracing algorithm that preserves common boundaries between regions”, CVGIP: Image Understanding, vol. 53, no. 3, pp. 313-321.

Lynch M., Robinson K., Ghita O., and Whelan P., 2004, “A performance characterisation in advanced data smoothing techniques”, In Irish Machine Vision and Image Processing, pp. 123-128.

Maintz J. B. A. and Viergever M. A., 1998, “A survey of medical image registration”, Medical Image Analysis, vol. 2, no. 1, pp. 1–36.

Maxwell B., Friedhoff R., and Smith C., 2008, “A bi-illuminant dichromatic reflection model for understanding images”, IEEE Conf. on Computer Vision and Pattern Recognition, pp. 1-8.

McFarlane N. and Schofield C., 1995, “Segmentation and tracking of piglets in images”, Machine Vision and Applications, vol. 8, no. 3, pp. 187-193.

Medioni G. G., 1999, “Detecting and Tracking Moving Objects for Video Surveillance”, In Proc. IEEE Intl. Conf. Computer Vision and Pattern Recognition, vol. 2, pp. 319-325.

Mittal A. and Paragios N., 2003, “Motion-Based Background Subtraction using Adaptive Kernel Density Estimation”, In Proc. of CVPR 2004, vol. 2, pp.302-309.

Miyatake T., Matsushima H., and Ejiri M., 1997, “Contour representation of binary images using run-type direction codes”, Machine Vision and Applications, vol. 9, pp. 193-200.

Morse B. S., 2000, “Lecture 12: Local image processing (smoothing)”, Brigham Young University.

Nadimi S. and Bhanu B., 2002, “Moving shadow detection using a physic-based approach”, In Proceedings of the 16th International Conference on Pattern Recognition, vol. 2, pp. 701-704.

Neri A., Colonnese S., Russo G., and Talone P., 1998, “Automatic moving objects and background separation”, Signal Processing, vol. 66, no. 2, pp. 219-232.

Newman W. M. and Sproull R. F., 1979, “Principles of Interactive Computer Graphics”, 2nd ed., McGraw-Hill, New York.

Niblack W., 1986, An Introduction to Digital Image Processing, Englewood Cliffs, N.J. Prentice Hall, pp. 115-116.

Oliver N. M., Rosario B., and Pentland A. P., 2000, “A Bayesian computer vision system for modeling human interactions”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 8, pp. 831–843.

Otsu N., 1979, “A threshold selection method from gray-level histograms”, IEEE Transactions on Systems, Man and Cybernetics, vol. 9, no. 1, pp. 62-66.

Paul J. B., 1986, “Surfaces in early range image understanding”, Ph.D. Thesis, the University of Michigan, MI.

Papamarkos N., Tzortzakis J., and Gatos B., 1996, “Determination of run-length smoothing values for document segmentation”, In the IEEE Proceedings of International Conference on Electronics, Circuits, and Systems, vol. 2, pp. 684-687.

Parker J. R., 1997, “Algorithms for Image Processing and Computer Vision”, John Wiley and Sons, New York, USA.

Piccardi M., 2004, “Background subtraction techniques: a review”, In Proc. IEEE Conference on Computer, http://www-staff.it.uts.edu.au/_massimo.

Porikli F. and Thornton J., 2005, “Shadow Flow: A Recursive Method to Learn Moving Cast Shadows”, In Proc. of Tenth IEEE International Conference on Computer Vision, vol. 1, pp. 891-898.

Prati A., Mikic I., Cucchiara R. and Trivedi M. M., 2003, “Detecting moving shadows: algorithms and evaluation”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, pp. 918-923.

Quek F. K. H., 2000, “An algorithm for the rapid computation of boundaries of run-length encoded regions”, Pattern Recognition, vol. 33, no.10, pp. 1637-1649.

Radke R. J. , Andra S., Al-Kofahi O., and Roysam B., 2005, “Image Change Detection Algorithms: A Systematic Survey”, IEEE Transactions on Image Processing, vol. 14, no. 3, pp. 294-307.

Ranganathan N., Mehrotra R., and Subramanian S., 1995. “A high speed systolic architecture for labeling connected components in an image”, IEEE Transactions on System, Man and Cybernetics, vol. 25, pp. 415-423.

Ren M., Yang J., and Sun H., 2002, “Tracing boundary contours in a binary image”, Image and Vision Computing, vol. 20, no. 2, pp. 125-131.

Ridder C., Munkelt O., and Kirchner H., 1995, “Adaptive background estimation and foreground detection using kalman filtering”, In Proceedings of Internat-ional Conference on recent Advances in Mechatronics, pp. 193–199.

Ridler T. and Calvard S., 1978, “Picture thresholding using an iterative selection method’’, IEEE Transactions on Systems, Man and Cyberntics, SMC-8, pp. 630–632.

Rosenfeld A., and Kak A. C., 1982, “Digital Picture Processing”, 2nd ed., Academic Press, New York, 1982.

Rosin, P., 2001, “Unimodal thresholding”, Pattern Recognition, vol. 34, no. 1, pp. 2083-2096.

Rosin P. L. and Ellis T., 1995, “Image difference threshold strategies and shadow detection”, In Proceedings of the 6th British Machine Vision Conference, UK, pp. 347-356.

Russell S. and Norvig P., 1995, “AI, ntelligence: A Modern Approach", Prentice Hall, Fig. 24.9, pg. 737.

Samet H., 1984, “The quadtree and related hierarchical data structures”, Computing Surveys, vol. 16, pp. 187-260.

Salvador E., Cavallaro A., and Ebrahimi T., 2003, “Spatio-temporal shadow segmentation and Tracking”, in Proceeding of SPIE’s Image and Video Communications and Processing, vol. 5022, pp. 389–400.

Salvador E., Cavallaro A., and Ebrahimi T., 2004, “Cast shadow segmentation using invariant color features”, Computer Vision and Image Understanding, vol. 95, no. 2, pp. 238-259.

Sechidis L., Patias, P., and Tsioukas, V., 2002, “Low-level tracking of multiple objects”, In Proc. of Photogrammetric Computer Vision, pp. 237-240.

Sezgin M. and Sankur B., 2004. “Survey over image thresholding techniques and quantitative performance evaluation”, Journal of Electronic Imaging, vol. 13, no. 1, pp. 146– 165.

Shih F. Y. and Wong W.-T., 1999, “A one-pass algorithm for local symmetry of contours from chain codes”, Pattern Recognition, vol. 32, no.7, pp. 1203-1210.

Shio A. and Sklansky J., 1991, “Segmentation of people in motion”, In Proceedings of IEEE Workshop on Visual Motion, pp. 325-332.

Shoushtarian B. and Bez H. E., 2005, “A practical adaptive approach for dynamic background subtraction using an invariant colour model and object tracking”, Pattern Recognition Letters, vol. 26, no. 1, pp. 5-26.

Solihin Y. and Leedham C. G., 1999, “Integral Ratio: A New Class of Global Thresholding Techniques for Handwriting Images”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 8, pp. 761-768.

Sonoda Y. and Ogata T., 1998, “Separation of moving objects and their shadows, and application to tracking of loci in the monitoring images" In Proceedings of IEEE Int. Conference on Signal Processing, pp. 1216-1264.

Sneath, P. and Sokal, R., 1973, “Numerical Taxonomy. The principle and practice of numerical classification, W.H. Freeman.

Spagnolo P., Leo M., Attolico G., and Distante A., 2003, “A supervised approach in background modelling for visual surveillance”, In Proceedings of the 4th International Conference on Audio- and Video-Based Biometric Person Authentication, pp. 592-599.

Spagnolo P., D’Orazio T., Leo M., and Distante A., 2006, “Moving object segmentation by background subtraction and temporal analysis”, Image and Vision Computing, vol. 24, no.5, pp. 411-423.

Stauder J., Mech R., and Ostermann J., 1999, “Detection of Moving Cast Shadows for Object Segentation”, IEEE Transactions on Multimedia, Vol. 1, no. 1, pp. 65-76.

Stauffer C. and Grimson W.E.L., 1999, “Adaptive background mixture models for real-time tracking”, In Proceedings of International Conference of Computer Vision and Pattern Recognition, Vol. 2, pp. 246-252.

Stauffer C. and Grimson W.E.L., 2000, “Learning patterns of activity using real-time tracking," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, pp. 747-757.

Suen C. Y., Nadal C., Legault R., Mai T. A., and Lam L., 1992, “Computer recognition of unconstrained handwritten numerals”, Proceedings of the IEEE, vol. 80, no. 7, pp. 1162-1180.

Suzuki S. and Abe K., 1983, “Border following algorithms for analyzing the topological structure of digitized binary images”, Technical Report of IEICE, PRL83-2, pp. 9-16.

Taxt T., Flynn P.J., and Jain A.K, 1989. “Segmentation of document images”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 12, pp. 1322-1329.

Thomas M., and Ngan K. N., 1998, “Automatic segmentation of moving objects for video object plane generation”, IEEE Transaction on Circuits and Systems for Video Technology, vol. 8, no. 5, pp. 525-538.

Tian J., Sun J., and Tang Y., 2009, “Tricolor Attenuation Model for Shadow Detection”, IEEE Transaction on Image Processing, vol. 18, no. 10, pp. 2355-2363.

Tian, Y-L., Lu M., and Hampapur A., 2005, “Robust and efficient foreground analysis for real-time video surveillance”, in Proceedings of IEEE Conf. on Computer Vision and Pattern Recognition, vol. 1, pp. 1182-1187.

Toyama K., Krumm J., Brumitt B., and Meyers B., 1999. “Wallflower: Principles and practice of background maintenance”, In Proc. International Conf. on Computer Vision, pp. 255–261.

Trimeche M., 2000, “Shape Representation for Image Indexing and Retrieval”, Master thesis, Tampere University, Finland.

Tsai D., 1995, “A fast thresholding selection procedure for multimodal and unimodal histograms”, Pattern Recognition Letters, vol. 16, pp. 653-666.

Umbaugh S. E.**,** 1998, "Computer Vision and Image Processing", Prentice-Hall International.

Wang C., Huang L., and Rosenfeld A., 1991, “Detecting clouds and cloud shadows on aerial photographs”, Pattern Recognition Letters, vol. 12, no. 1, pp. 55-64.

Wang K., Chia T., ZEN Chen Z., and Lou D., 2003. “Parallel execution of a connected component labeling operation on a linear array architecture”, Journal of Information Science and Engineering, vol. 19, pp. 353-370.

Wang D. and Srihari S., 1989, “Classification of newspaper image blocks using texture analysis”, Computer Vision, Graphics, and Image Processing, vol. 47, pp. 327-352.

Wixen L. and Hansen M., 1999, “Detecting salient motion by accumulating directional-consistent flow”, in Proceedings of International Conference on Computer Vision, vol. II, pp. 797-804.

Wong K. Y., Casey R. G., and Wahl F. M., 1982, “Document analysis system”, IBM Journal of Research Development, vol. 26, no. 6, pp. 647-656.

Wren C., Azarbayejani A. and Darrell T., 1997, “Pfinder: Real-time tracking of the human body”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 780-785.

Xiao Y., Zou J. J., and Yan H., 2001, “An adaptive split-and-merge method for binary image contour data compression”, Pattern Recognition Letters, vol. 22, no. 3-4, pp. 299-307.

Xiuwen L., Wang D. L., and Ramirez J. R., 2000, “Boundary detection by contextual non-linear smoothing”, Pattern Recognition, vol. 33, no.2, pp. 263-280.

Yager R., 1979, “On the measure of fuzziness and negation. Part I: Membership in the unit interval”, Int. Journal of Gen. Systems, vol. 5, pp. 221-229.

Yang Y-H, Levine M. D., 1992, “The background primal sketch: an approach for tracking moving objects”, Machine Vision and Applications, vol. 5, pp. 17-34.

Yokoi S., Toriwaki J., and Fukumura A., 1973, “An analysis of topological properties of digitized binary pictures using local features”, Transactions of the Institue of Electronics, Information and Communication Engineers, vol. 11, pp. 662-669.

Yoneyama A., Yeh C. H., and Kuo C. -C. J., 2003, “Moving cast shadow elimination for robust vehicle extraction based on 2d joint vehicle/shadow models”, In Proc. of IEEE Conf. on Advanced Video and Signal Based Surveillance, pp. 229-236.

Yoshinari K., and Michihito M., 1996, “A human motion estimation method using 3-successive video frames”, In Proceedings of Intl. Conference on Virtual Systems and Multimedia, pp. 135-140.

Yu D. and Yan H., 1997, “An efficient algorithm for smoothing linearization and detection of structural feature points of binary image contours”, Pattern Recognition, vol. 30, no. 1, pp. 57-69.

Yuan J. and Suen C. Y., 1995, “An optimal O(*n*) algorithm for identifying line segments from a sequence of chain codes”, Pattern Recognition, vol. 28, no. 5, pp. 635-646.

Zha Y., Yang Y., Zhang M., and Bi D., 2007, “Moving Cast Shadow Detection by Energy Minimization”, IEEE Conf. on Image and Graphics, pp. 235-240.

Zhang W., Fang X., and Yang X., 2007, “Moving cast shadows detection using ratio edge,” IEEE Transactions on Multimedia, vol. 9, no. 6, pp. 1202-1214.

Zitova B. and Flusser J., 2003, “Image registration methods: a survey”, Image and Vision Computing, vol. 21, pp. 977-1000.

Zingaretti P., Gasparroni M., and Vecci L., 1998, “Fast chain coding of region boundaries”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 4, pp. 780-785.

Zivkovic Z. and van der Heijden F., 2006, “Efficient adaptive density estimation per image pixel for the task of background subtraction”, Pattern Recognition Letters, vol. 27, no. 7, pp. 773–780.