

國立交通大學

電機學院 電子與光電學程

碩士論文

適用於靜態背景視訊並以型態學處理物件邊緣之
即時視訊切割

Real-time Segmentation of Video with Stationary Background Based on
Morphological Edge Processing

研究生：黃奕善

指導教授：林大衛 教授

中華民國九十六年六月

適用於靜態背景視訊並以型態學處理物件邊緣之
即時視訊切割

Real-time Segmentation of Video with Stationary Background Based on
Morphological Edge Processing

研究生：黃奕善

Student: I-Shan Huang

指導教授：林大衛

Advisor: Dr. David W. Lin



A Thesis

Submitted to College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science
in

Electronics and Electro-Optical Engineering

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年六月

適用於靜態背景視訊並以型態學處理物件邊緣 之即時視訊切割

研究生：黃奕善

指導教授：林大衛 教授

國立交通大學 電機學院 電子與光電學程碩士班

摘要



在本論文中，我們設計並實現一個在個人電腦上的視訊影像切割系統。此系統可以應用於靜態背景的視訊電話及視訊會議。

此系統的基本概念是利用物件的邊緣銳化來精準得到移動的物件。首先，我們使用一個兩級的雜訊估計方法來估計攝影機的雜訊，並且把此結果拿來當做往後參數調整的參考。為了取得一個較佳的物件遮罩，我們觀察六張連續畫面的變化情形來取得一個初步的前景以及利用連續兩張畫面的差異選擇一適合的臨界值將移動的物件萃取出來。接著，我們利用 Canny edge detector 偵測出整張畫面的邊界資訊，利用影像變動偵測所得之遮罩及前一張畫面所求得之物件遮罩可以將屬於背景的邊界移除，然後使用收縮的技巧來取得一個粗略的物件遮罩，並利用此資訊來取得移動物件最外圍之邊界資訊。為了得到更精確的物件遮罩，我們使用 Dijkstra 所提之最短路徑搜尋演算法，將物件的邊緣連接並利用此封閉的物件輪廓將移動的物件給萃取出來。最後的模擬結果顯示我們所提的方法，切割出來的物體邊界是相當精確的，且在經過我們的加速之後，整個系統能快速且精準的切割出移動的物件。

在 1.733-GHz CPU 及 1024-MB RAM 的個人電腦上且攝影機不移動時，目前的執行速度是每秒約四張 CIF Frame 及每秒約二十張 QCIF Frame，在格式影片的應用上，我們有提出一簡化的方法，則每秒約十二張。

Real-Time Segmentation of Video With Stationary Background Based on Morphological Edge Processing

Student: I-Shan Huang

Advisor: Dr. David W. Lin

Degree Program of Electrical and Computer Engineering
National Chiao Tung University

Abstract

In the thesis, we consider the design and implementation of video segmentation on a personal computer. The system can be applied on video conference and videophone with stationary background.

The basic idea of the system is a graph-based edge linking technique. At first, we adopt a two staged method for camera noise estimation and those thresholds are adjusted based on the estimated camera noise. To get the change detection mask, we consider six consecutive frames as time of observation to estimate the background and select a suitable threshold to estimate the foreground by two consecutive frames. Next, we use Canny edge detector to get the edge information of entire frame. We use the change detection mask and the moving object mask of previous frame to remove the edges of background. Then, we shrink the object mask to edge map. To refine the object mask, we use Dijkstra's shortest path search algorithm to link the boundary of moving object and extract the object by the closed contour. Simulation results show

that our method can give correct segmentation results. After optimization, the proposed segmentation system can get the moving object accurately and quickly.

With a Intel Pentium M 1.733 GHz CPU and 1024-MB RAM, the system can achieve 20 QCIF frames per second and 4 CIF frames per second. We also propose a simple method and it can achieve 12 CIF frames per second.



誌 謝

本篇論文的產生，最感謝的是我的指導教授—林大衛老師。身為一位在職研究生要兼顧研究學習及工作本來是件不容易的事，但老師給予我多方面的協助與鼓勵，讓我在研究學習中能夠順利進行，使我在研究學習及工作中得以兼顧而能完成本篇論文。老師除給了我完善的專業知識訓練外，在論文實作部份，老師更培養了我認真踏實的的研究態度，讓我獲益良多。

實驗室完善的資源，讓我可以順利的克服許多實作上所遇的的困難。感謝學弟介遠、政達，在你們的討論與幫助下，讓我在工作中亦保有快樂的研究生生涯。



最後，要特別感謝我的爸媽。他們給了我精神上的支持與鼓勵，陪伴我度過所有的艱難。在此，我要將我的論文獻給所有關心與幫助我的人。

黃奕善

民國九十六年七月於新竹

Contents

1	Introduction	1
2	Overview of Video Segmentation	3
2.1	Meaning of Segmentation	3
2.2	Basic Procedure of Segmentation	3
2.3	Change Detection-Based Technique	5
2.4	Background Subtraction-Based Technique	6
2.5	Graph-Based Edge Linking Technique	10
2.6	Summary	12
3	The Proposed Segmentation Method	14
3.1	System Overview	14
3.2	Two-Stage Noise Estimation	14
3.2.1	Influence of Noise	14
3.2.2	Motivation for Noise Estimation	16
3.2.3	Camera Noise Model	16
3.2.4	Procedure for Two-Stage Noise Estimation	16
3.3	Temporary Moving Object Mask	22
3.3.1	Foreground Estimation	22
3.3.2	Mathematical Morphological Operator	23
3.3.3	Background Estimation	23
3.3.4	Obtaining Initial Object Mask	25
3.3.5	Refining Initial Object Mask	25

3.4	Edge Linking for Moving Object	29
3.4.1	Find the Candidate of Edge Map	32
3.4.2	Overview of Shortest Path Algorithm	32
3.4.3	Edge-Link Method	35
3.5	Post-Processing	35
3.6	Temporal Filtering	37
3.7	Conclusion	39
4	Optimized Implementation on Personal Computer	42
4.1	Optimization of Algorithm	42
4.2	Overview of Intel's MMX Technology	44
4.2.1	The MMX Architecture [16], [17], [18]	44
4.2.2	The MMX Instruction Set [18]	47
4.3	Code Acceleration	52
4.3.1	Labeling Function Optimization	52
4.3.2	Edgeline Function Optimization	54
4.3.3	Post_processing Function Optimization	56
4.3.4	Erosion and Dilation Function Optimization	58
4.4	Conclusion	60
5	Simulation Results	63
5.1	Segmented Object Masks	63
5.1.1	Segmentation Results	63
5.1.2	Performance Evaluation	64
5.2	Run-time Analysis	64
5.3	Complexity Analysis	69
6	Integration of System on Personal Computer	76
6.1	Video Capturing	77
6.1.1	Video for Windows	77
6.1.2	AVI Format	77

6.1.3	Implementation of Capture	78
6.2	Result Display	79
6.3	Graphed User Interface	79
6.4	Conclusion	80
7	Conclusion and Future Work	82



List of Tables

4.1	Profile and Run Time Comparison of Claire Sequence (CIF)	43
4.2	Run Time of Optimization of Labeling Function per Frame	44
4.3	MMX Instruction Set Summary (from [19])	49
4.4	MMX Instruction Set Summary (from [19])	50
4.5	Run Time of Optimization of Labeling Function per Frame	54
4.6	Run Time of Optimization of Edgeline Function per Frame	56
4.7	Run Time of Optimization of Post-process Function per Frame	58
4.8	Run Time of Optimization of Erosion and Dilation Function per Frame	58
4.9	Profile and Run Time Comparison of Claire Sequence (CIF) in Debug Mode	61
4.10	Run Time Comparison of Claire Sequence (CIF) in Debug Mode and Release Mode	62
5.1	Profile and Run Time Comparison of Akiyo Sequence (CIF) in Debug Mode	71
5.2	Profile and Run Time Comparison of LabI Sequence (CIF) in Debug Mode	72
5.3	Run Time and Frame Rate of Different Sequences with CIF or QCIF Formats in Release Mode	73
5.4	Run Time and Frame Rate of Different Sequences use Simple Segmentation Method in Release Mode	73
5.5	Comparison of Run-time between Different Algorithms	74
5.6	Complexity of Major Algorithms in One CIF Frame of Claire Sequence	75

List of Figures

2.1	A basic segmentation system (from [8]).	4
2.2	Background subtraction-based segmentation system (from [4]).	8
2.3	Initial object mask generation (from [4]).	9
2.4	Graph-based edge linking segmentation system (from [5]).	11
2.5	Correction of a boundary segment. (a) Enlarged portion of the wrong boundary. (b) Corresponding binary model points. (c) Weights assigned for Dijkstra’s algorithm. (From [5]).	13
3.1	The block diagram of proposed method.	15
3.2	A thresholded frame difference map of the Mother-and-Daughter sequence (from [8]).	17
3.3	A thresholded frame difference map of the Akiyo sequence (from [8]). . .	17
3.4	A frame difference map of Mother-and-Daughter sequence (from [8]). . .	19
3.5	Four masks of directional sums (from [8]).	19
3.6	Mother-and-Daughter sequence (from [8]).	19
3.7	Claire sequence (from [8]).	20
3.8	Noise estimation of Mother-and-Daughter sequence (from [8]).	20
3.9	Noise estimation of Claire sequence (from [8]).	21
3.10	Noise estimation of Mother-and-Daughter sequence for the two-stage method (from [8]).	21
3.11	Noise estimation of Claire sequence for the two-stage method (from [8]).	22
3.12	Threshold frame difference map of frame 27 in Claire sequence.	24

3.13	Threshold frame difference map of frame 27 in Claire sequence after closing operation.	24
3.14	The original frame 60 in Claire sequence.	26
3.15	Threshold frame difference map after foreground estimation of frame 60 in claire sequence.	26
3.16	Threshold frame difference map after background estimation of frame 60 in Claire sequence.	27
3.17	The OR map of Fig 3.15 and Fig 3.16.	27
3.18	Threshold frame difference map after remove small region of frame 60 in Claire sequence.	28
3.19	Threshold frame difference map after fill-in of frame 60 in Claire sequence	28
3.20	Edge map of frame 60 in Claire sequence.	30
3.21	Edge map after opening operator of frame 60 in Claire sequence.	30
3.22	Edge map after removing background edges of frame 60 in Claire sequence.	30
3.23	Final object edge map of frame 60 in Claire sequence.	31
3.24	Final object mask of frame 60 in Claire sequence.	31
3.25	The edge map of frame 49 after Canny operation in Mother-and-Daughter sequence.	33
3.26	The edge map of frame 49 after removing background edges in Mother-and-Daughter sequence.	33
3.27	The outmost edge map of frame 49 in Mother-and-Daughter sequence.	34
3.28	The weighted edge map of frame 49 in Mother-and-Daughter sequence.	34
3.29	Flowchart of edge-link method.	36
3.30	The edge map of frame 49 after edge-link in Mother-and-Daughter sequence.	36
3.31	The object mask of frame 49 after padded with edge pixels in Mother-and-Daughter sequence.	38
3.32	The original frame 49 in Mother-and-Daughter sequence.	38
3.33	The moving object of frame 49 in Mother-and-Daughter sequence.	38

3.34	Flowchart of temporal filter.	39
3.35	Segmentation of the Claire sequence. (a) The original frame 29. (b) The original frame 30. (c) The original frame 31. (d) Frame 29 without temporal filter. (e) Frame 30 without temporal filter. (g) Frame 31 without temporal filter. (g) Frame 29 with temporal filter. (h) Frame 30 with temporal filter. (i) Frame 31 with temporal filter.	40
4.1	The object mask after applying down-sample.	45
4.2	The object mask after applying labeling and up-sample.	45
4.3	MMX packed data types (from [16]).	47
4.4	MMX register set (from [19]).	48
4.5	PACKSSDW instruction operation using 64-bit operands (from [18]). . .	51
4.6	Section of code that we would like to optimize with MMX.	53
4.7	Code in Fig. 4.6 after optimization using MMX instructions.	53
4.8	The section of original code with malloc function.	55
4.9	The section of revised code with calloc function.	55
4.10	The section of original code without break instruction.	55
4.11	The section of revised code with break instruction.	56
4.12	Section of code that we would like to optimize with MMX.	57
4.13	Code in Fig. 4.12 after optimization using MMX instructions.	57
4.14	The section of original code without continue instruction.	59
4.15	The section of original code with continue instruction.	59
5.1	Segmentation of the Mother-and-Daughter sequence. (a) Frame 48. (b) Frame 156. (c) Frame 234. (d) Segmented object in frame 48. (e) Segmented object in frame 156. (f) Segmented object in frame 234.	65
5.2	Segmentation of the Claire sequence. (a) Frame 16. (b) Frame 63. (c) Frame 104. (d) Segmented object in Frame 16. (e) Segmented object in Frame 63. (f) Segmented object in Frame 104.	66

5.3	Segmentation of the Akiyo sequence. (a) Frame 24. (b) Frame 60. (c) Frame 141. (d) Segmented object in Frame 24. (e) Segmented object in Frame 60. (f) Segmented object in Frame 141.	67
5.4	Segmentation of the LabI sequence. (a) Frame 18. (b) Frame 19. (c) Frame 20. (d) Segmented object in frame 18. (e) Segmented object in frame 19. (f) Segmented object in frame 20.	68
5.5	Fractional agreement between the segmented foreground object of the Akiyo sequence and the reference mask.	68
5.6	Flowchart of simple segmentation method.	70
6.1	System block diagram (from [8]).	76
6.2	Circumstance of implementation (from [8]).	77
6.3	AVI header (from [8]).	78
6.4	Related code for creating a capture window.	78
6.5	Related code for parameter modification.	79
6.6	Related code for capture operation.	79
6.7	Related code for displaying.	80
6.8	Application program interface.	81

Chapter 1

Introduction

We consider the design and implementation of video segmentation system on a personal computer. The intended application is PC-based videoconference and videophone system. We assume the camera is fixed.

Video segmentation is one of the fundamental technologies for content-based real-time MPEG-4 camera system. For real-time requirement, the segmentation algorithm should be fast and accurate. The main objective in segmentation is to partition the data into meaningful and independent regions. According to the type of data, we can classify the processes into image segmentation and video segmentation. Here, we focus on the video segmentation. The general schemes for video segmentation can be seen as the following steps: pre-processing, feature extracting, decision, and post-processing. In this thesis, a method based on change detection and graph-based edge linking is proposed.

The basic idea of the system is a graph-based edge linking technique. The moving objects of current frame can be roughly obtained by the change detection technique. It requires statistical modeling of the background noise. To begin, we estimate the variance of the background noise by finding the difference of consecutive frames. Then we obtain a compact image region by a fill-in procedure employing the edge map from employing the Canny method. However, typical edge detection algorithms, including the Canny method [1], do not yield closed region contours or fully connected curves at object boundaries. To help object segmentation, we employ a shortest-path algorithm to find and connect the boundary edges and the favorite algorithm is Dijkstra's algorithm [2].

After linking edge, we could obtain a closed object boundary. The postprocessing is to eliminate the redundant region and smooth the object boundary. To smooth the boundary, morphological operations are applied. At last, the object alpha map is generated after this process.

The technologies described as above deal with the video in spatial domain in addition to change detection. In temporal domain, we use the temporal filter. The simplest way to judge whether the value of a pixel is background is to check the alpha map at this location. Our idea is to record the weight of variability of consecutive alpha frames. If the weight is greater than the threshold, the pixel is considered that stationary.

In many steps of our algorithm, we need a threshold to make decisions. In many situations, the threshold is adjusted to counteract the influence of noise and therefore we estimate the camera noise at first and the adjustment of threshold is based on camera noise.

We propose an automatic segmentation system which include change detection, background registration, and graph-based edge linking techniques. At first, we modify and integrate these techniques to get the most accurate object mask. Second, we improve the system to achieve the real-time requirement. Finally, we integrate our segmentation system on personal computer to test natural video sequence.

This thesis is organized as follows. Chapter 2 is an overview of popular video segmentation techniques. Chapter 3 gives a detailed description of proposed segmentation method. Chapter 4 discuss the optimization of proposed video segmentation method. Chapter 5 shows the simulation result. Chapter 6 shows the integration of the proposed segmentation method on the personal computer. Chapter 7 contains the conclusion and future work.

Chapter 2

Overview of Video Segmentation

2.1 Meaning of Segmentation

The main objective in segmentation is to partition the data into meaningful and independent regions. According to the type of data, we can classify the segmentation processes into image segmentation and video segmentation. In the case of image segmentation, the problem is two-dimensional in nature, while in case of video segmentation, in addition to the two-dimensional information we can also handle the problem with the aid of motion information.

The algorithm of segmentation depends to a large extent on the application and the data that are used. In one application, partition into ten regions may be ideal but for an other application partition into two regions may be desired. In the ideal case, we may develop a best algorithm for each application but it likely come with very high complexity. In practice, we often develop an algorithm best suited to specific applications, not to general situations.

2.2 Basic Procedure of Segmentation

In [8], the author mention that a general scheme for segmentation can be seen as composed of the following steps: pre-processing, feature extraction, decision, and post-processing, as illustrated Figure 2.1.

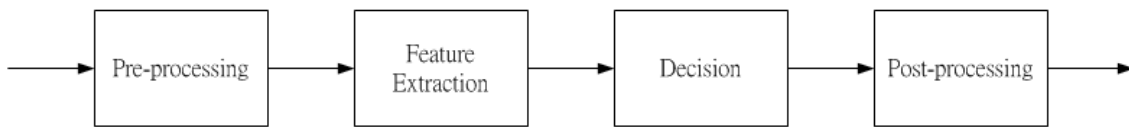


Figure 2.1: A basic segmentation system (from [8]).

1. Pre-processing:

The original data may include a high amount of information but most of it is irrelevant to our application and sometimes influences our decision. In the stage, we remove the irrelevant information and keep the desired data.

2. Feature extraction:

Depending on various segmentation algorithms, we need some specific features to achieve our goals. In this stage, we extract the desired features from original data.

3. Decision:

The values of extracted features are usually more than two values and therefore we need to find a threshold to classify those data. In the stage, we can use many strategies to make a desired threshold set to find out the regions which meet our goal.

4. Post-processing:

Although we can use many decision strategies to make a best threshold set, we may still make wrong decisions in some special regions. In the stage, we will try to correct those improper regions.

In the following sections, we introduce some popular segmentation techniques.

2.3 Change Detection-Based Technique

It is possible to extract the independently moving objects from background by calculating frame difference between consecutive frames. This technique is usually named change detection. The main advantage of change detection is that it has relatively low computational complexity and can extract the moving objects roughly. Many video segmentation algorithms employ this technique for simplicity.

Change detection algorithms usually start with the gray value difference map between the two frames considered. The local sum (or mean) of absolute difference is computed inside a small measurement window which slides over the difference map. At each location, this local sum of absolute differences is compared against a threshold. Whenever this threshold is exceeded, the center pixel of the current window location is marked as changed. The key issue of this technique is how to decide the threshold.

The algorithm in [3] provides some statistical methods to decide the threshold. It starts by computing the gray level difference image $D = \{d_k\}$, with $d_k = y_1(k) - y_2(k)$, between two considered pictures $Y_1 = \{y_1(k)\}$ and $Y_2 = \{y_2(k)\}$. The index k denotes the pixel location on the image grid. Under the hypothesis that no change occurs at location k , the corresponding difference d_k obeys a zero mean Gaussian distribution $N(0, \sigma)$ with variance σ^2 , that is,

$$p(d_k|H_0) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{d_k^2}{2\sigma^2}\right\}. \quad (2.1)$$

Since the camera noise is uncorrelated between different frames, the variance σ^2 is equal to twice the variance of the assumed Gaussian camera noise distribution. H_0 denotes the null hypothesis, i.e., the hypothesis that there is no change at pixel k . The unknown parameter σ can be estimated offline for the used camera system, or recursively online.

In order to make the detection more reliable, a region may be used to evaluate the difference d_i instead of only a single pixel. One may thus compute the local sum Δ_i^2 of $(\frac{d_k}{\sigma})^2$ inside a small sliding window w_i , with i denoting the center pixel of the window.

The local sum Δ_i^2 is proportional to the sample mean of $(\frac{d_k}{\sigma})^2$ as computed inside the window. Under the assumption that no change occurs inside the window when centered at location i , the normalized difference $(\frac{d_k}{\sigma})$ obeys a zero-mean Gaussian distribution

$N(0, 1)$ with variance 1. Thus, the sum Δ_i^2 obeys a χ^2 -distribution with as many degrees of freedom as there are pixels inside the window. With the distribution $p(\Delta^2|H_0)$ known, the decision between “changed” and “unchanged” can be arrived at by a significance test. For this purpose, one may usually specify a significance level α and compute a corresponding threshold t_α according to

$$\alpha = Prob(\Delta_i^2 > t_\alpha | H_0). \quad (2.2)$$

The statistic Δ_i^2 is now evaluated at each location i on the image grid, and whenever it exceeds t_α , the corresponding pixel is marked as changed, otherwise as unchanged.

The method given above exhibits some shortcomings. First, there are inevitably decision errors. Typically, these errors appear as small isolated spots inside correctly labeled regions. Another drawback is that in some critical image areas the boundaries between differently classified regions tend to be somewhat irregular. To avoid these shortcomings, the MAP (maximum a posteriori) criterion can be used to get a better change mask. That is to say, one may try to find the change mask $Q = q_k$ which maximizes the a posteriori density $p(Q|D)$, where D is the given difference image. The label q_k at location k can take either the value u for “unchanged” or c for “changed”. After a series of deduction, we can get the final decision rule as

$$d_k^2 \begin{matrix} > \\ < \end{matrix} \begin{matrix} u \\ c \end{matrix} 2 \frac{\sigma_c^2 \sigma^2}{\sigma_c^2 - \sigma^2} \left(\ln \frac{\sigma_c}{\sigma} + (v_B(c) - v_B(u))B + (v_C(c) - v_C(u))C \right). \quad (2.3)$$

where B is a positive cost term of each horizontally or vertically oriented border pixel pair and C is that of diagonally pair, and $v_B(q_k)$ and $v_C(q_k)$ denote the number of inhomogeneous cliques to which pixel k belongs when its label is q_k .

2.4 Background Subtraction-Based Technique

The idea of background subtraction is to subtract the current image from the still background, which is acquired before the objects move in. After subtraction, only non-stationary or new object are left. The most straightforward way to separate background is to apply a simple difference and threshold method.

An example of background subtraction-based technique is proposed in [4]. This segmentation system consists of five major steps as shown in Figure 2.2.

1. Frame difference:

The frame difference mask is generated simply by thresholding the frame difference. This data is sent to the background registration step where the reliable background is constructed from the accumulated information of several frame difference masks. Since the accumulated frame difference mask are used in the final decision for a reliable background, no filtering or boundary relaxation is applied on the frame difference.

The changed detection technique as show in last section is used to obtain the threshold value.

2. Background registration:

The goal of background registration is to construct a reliable background information from the video sequence. In this application, it needs a reliable background information for change detection.

The history of frame difference mask is considered in constructing and updating the background buffer. A stationary map is maintained for this purpose. If a pixel is marked as changing in the frame difference mask, the corresponding value in the stationary map is cleared to zero; otherwise, if the pixel is stationary, the corresponding value is incremented by one. The values in the stationary map indicate that the corresponding pixel has not been changing for how many consecutive frames. If a pixel is stationary for the past several frames, then the probability is high that it belongs to the background region. Therefore, if the value in the stationary map exceeds a predefined value, then the pixel value in the current frame is copied to the corresponding pixel in the background buffer.

A background registration mask is also changed in this process. The value in the background registration mask indicates that whether the background information of the corresponding pixel exists or not. If a new pixel value is added into the

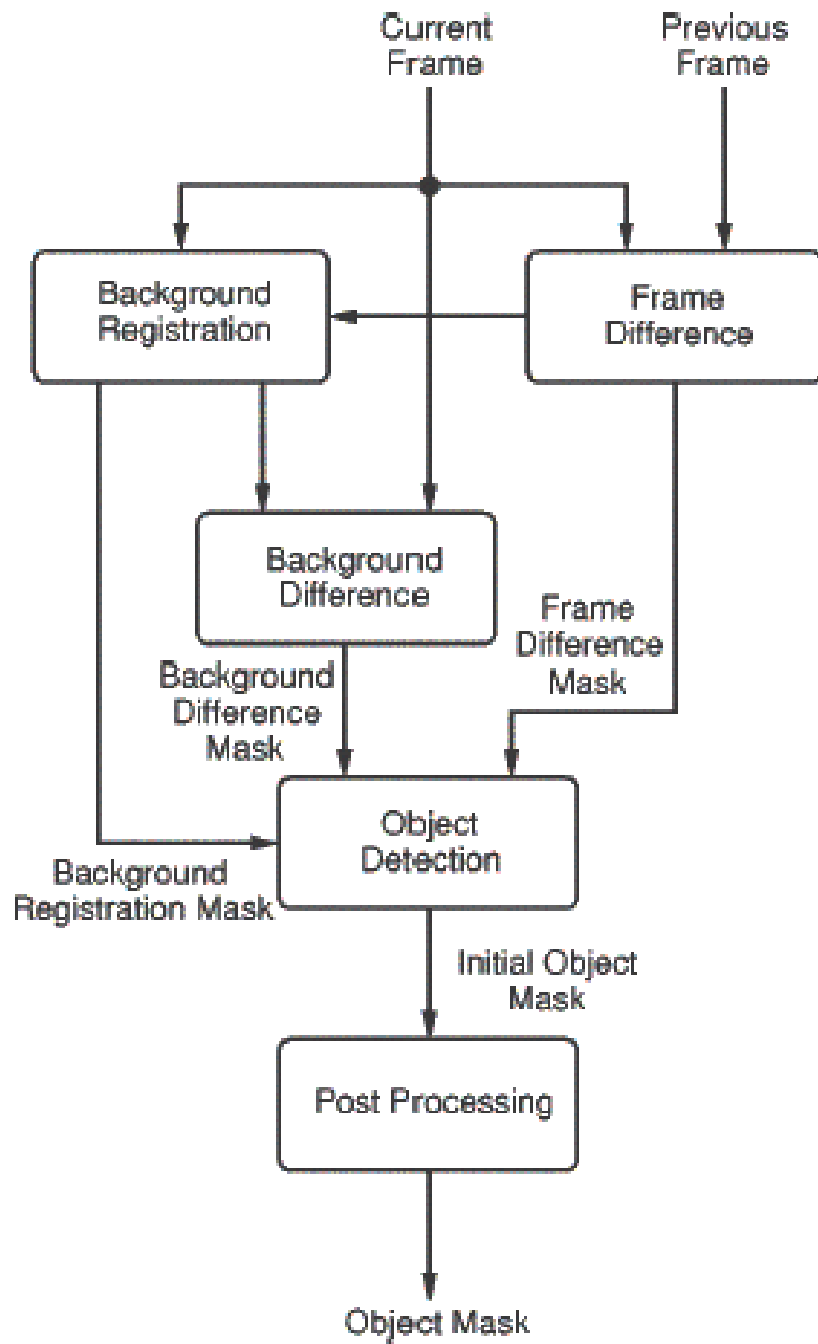


Figure 2.2: Background subtraction-based segmentation system (from [4]).

background buffer, the corresponding value in the background registration mask is changed from nonexistent to existing.

3. Background difference:

This step generates a background difference mask by thresholding the difference between the current frame and the background information stored in the background buffer. This step is very similar to the generation of frame difference mask.

4. Object detection:

The object detection step generates the initial object mask from the frame difference mask and the background difference mask. The background registration mask, frame difference mask, and background difference mask of each pixel are required information. Figure 2.3 lists the criteria for object detection, where BD means the absolute value of difference between the current frame and the background information stored in the background buffer, FD is the absolute value of frame difference, and the OM field indicates that whether or not the pixel is included in the object mask. TH_{BD} and TH_{FD} are the threshold values for generating the background difference mask and frame difference mask, respectively. If BD is greater than TH_{BD} , the pixel would be judged belong to moving object. If the BD is not exist, it would depend on the FD to judge the pixel change or unchange. And the condition is the same as change detection technique.

5. Post-processing:

Index	Background Difference	Frame Difference	Region Description	OM
1	N/A	$ FD > TH_{FD}$	Moving	Yes
2	N/A	$ FD \leq TH_{FD}$	Stationary	No
3	$ BD > TH_{BD}$	$ FD > TH_{FD}$	Moving Object	Yes
4	$ BD \leq TH_{BD}$	$ FD \leq TH_{FD}$	Background	No
5	$ BD > TH_{BD}$	$ FD \leq TH_{FD}$	Still Object	Yes
6	$ BD \leq TH_{BD}$	$ FD > TH_{FD}$	Uncovered Background	No

Figure 2.3: Initial object mask generation (from [4]).

After the object detection step, an initial object mask is generated. However, due to the camera noise and irregular object motion, there exist some noise regions in the initial object mask. The approach to eliminate the noise region relies on an observation that the area of noise regions tend to be smaller than the area of the object. Regions with area smaller than a threshold value are removed from the object mask. In this way, the object shape information is preserved while smaller noise regions are removed. After removing noise regions, a close and an open operation with a 3×3 structuring element are applied on the object mask.

2.5 Graph-Based Edge Linking Technique

In [5],[6],[7], authors proposed an automatic video segmentation algorithm based on graph-based edge linking technique. The algorithms link up the boundaries of the moving object to extract the moving object. In the following we will show a system proposed in [5] in which the edge linking technique is used. The system incorporates edge tracking has been proposed. A dense optical flow field and a global affine motion model has been computed to derive the motion mask. After the initial binary model for the interest is derived, the object tracker matches the model against subsequent frames in the sequence and updates the model. The basic idea of the algorithm is shown in Figure 2.4.

In the block of Video Object Plane (VOP) Extraction, the shortest path algorithm has been applied in order to correct the incorrect object boundaries resulting from the use of the filling-in procedure. Each wrong boundary segment is corrected separately. After removing one of those, there will be a gap in the otherwise closed contour, as illustrated in Figure 2.5(a).

Naturally, the contour between A and B should coincide with binary model points, which are showed in Figure 2.5(b). Hence, a small weight or distance d_0 is assigned to these model points. The weight is also set to d_0 for all pixels along the four frame boundaries, because in many situations they form part of the VOP contour. All remaining pixels are less likely to belong to the VOP boundary and a weight d_1 with $d_1 > d_0$ is assigned to them. Furthermore, the goal is to close the gap between A and B with the

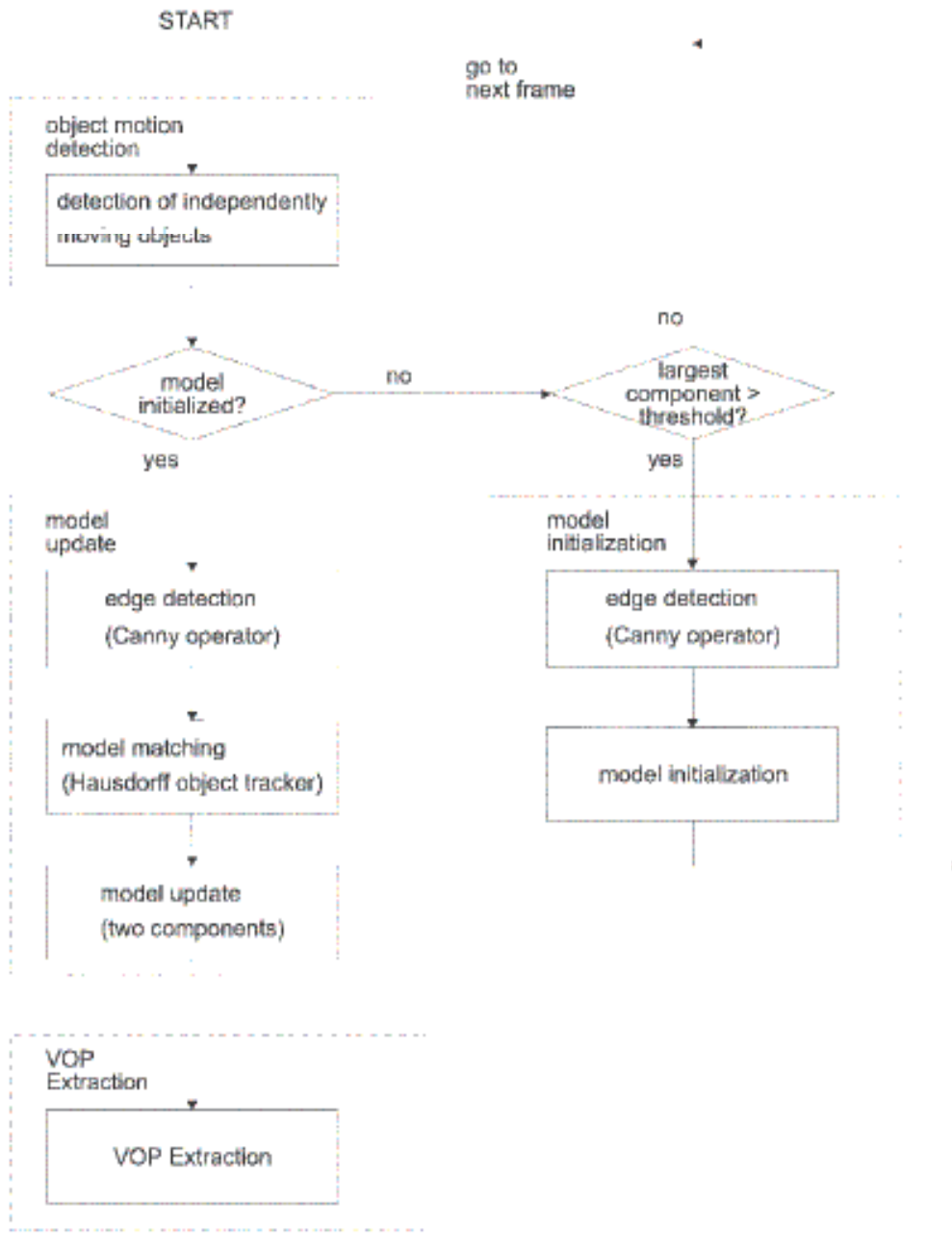


Figure 2.4: Graph-based edge linking segmentation system (from [5]).

corrected boundary segment. Therefore, all pixels that already belong to the VOP contour after removing the wrong boundary segment must be excluded. This is accomplished by setting the corresponding weights to infinity. To summarize, pixels that already belong to the VOP boundary must be excluded from the search by assigning infinity or a suitably large distance to them. The distance is set to d_0 both for object model points that are not yet VOP boundary pixels and for pixels along the four frame boundaries. All other pixels have a weight of d_1 . Figure 2.5(c) visualizes the weights assigned to the pixels in example.

This technique has to link the boundaries of moving object and get the closed contour. It can extract the moving objects for accurate locating of moving object boundaries. But this scheme is quite computationally expensive.

2.6 Summary

All techniques described above can obtain a desired object mask. Here, we explain the reason why we employ these techniques. In our application, the camera is fixed. For real time requirement, we employ the change detection-based technique to get the moving objects roughly because it can avoid complex computation. The object masks obtained by change detection-based technique are hard to resist the camera noise and the accuracy of object mask is hard to keep for the entire sequence. So we employ graph-based edge linking technique to get the closed contour of moving object and use it to refine the object masks.

However, since the behavior and characteristics of the moving objects differ significantly, the quality of segmentation result depends strongly on background noise, object motion, and the contrast between the object and the background. Reliable and consistent object information from the changing part of the scene is not easy to maintain for the entire sequence, we concentrate on the stationary background where the characteristics are well known and more reliable. So we employ the background registration technique in temporal domain to smooth the boundaries of moving object.

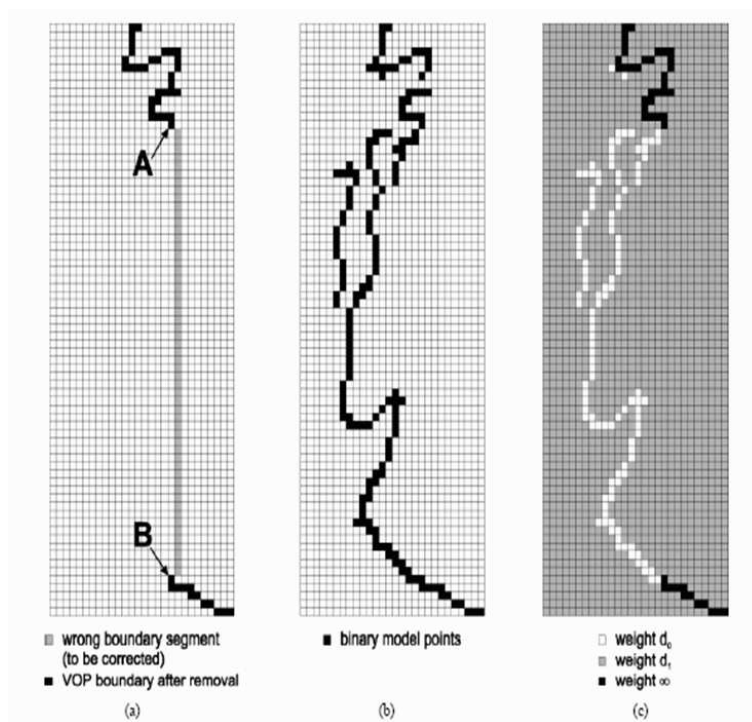


Figure 2.5: Correction of a boundary segment. (a) Enlarged portion of the wrong boundary. (b) Corresponding binary model points. (c) Weights assigned for Dijkstra's algorithm. (From [5]).

Chapter 3

The Proposed Segmentation Method

3.1 System Overview

Our segmentation system employs a graph-based edge linking scheme. The block diagram is shown in Figure 3.1. To start, we estimate the camera noise and the following thresholds are decided according to estimated camera noise. We use the foreground estimation and background estimation to generate the object mask. The edge map can be obtained by Canny operator in current frame. We can remove the edges of background and shrinks object mask to edge map. According to shortest path algorithm, we can get the closed contour of moving object. In the step of postprocessing we refine the object mask and get accurate object mask. The boundaries of moving object can be smoothed by temporal filter.

In this system, most of these modules are used for processing each frame. The details of this system are discussed in the following sections.

3.2 Two-Stage Noise Estimation

3.2.1 Influence of Noise

In this system, the image is captured by camera and then we get the initial image from the output of camera. In the process of capturing, the image may suffer from camera

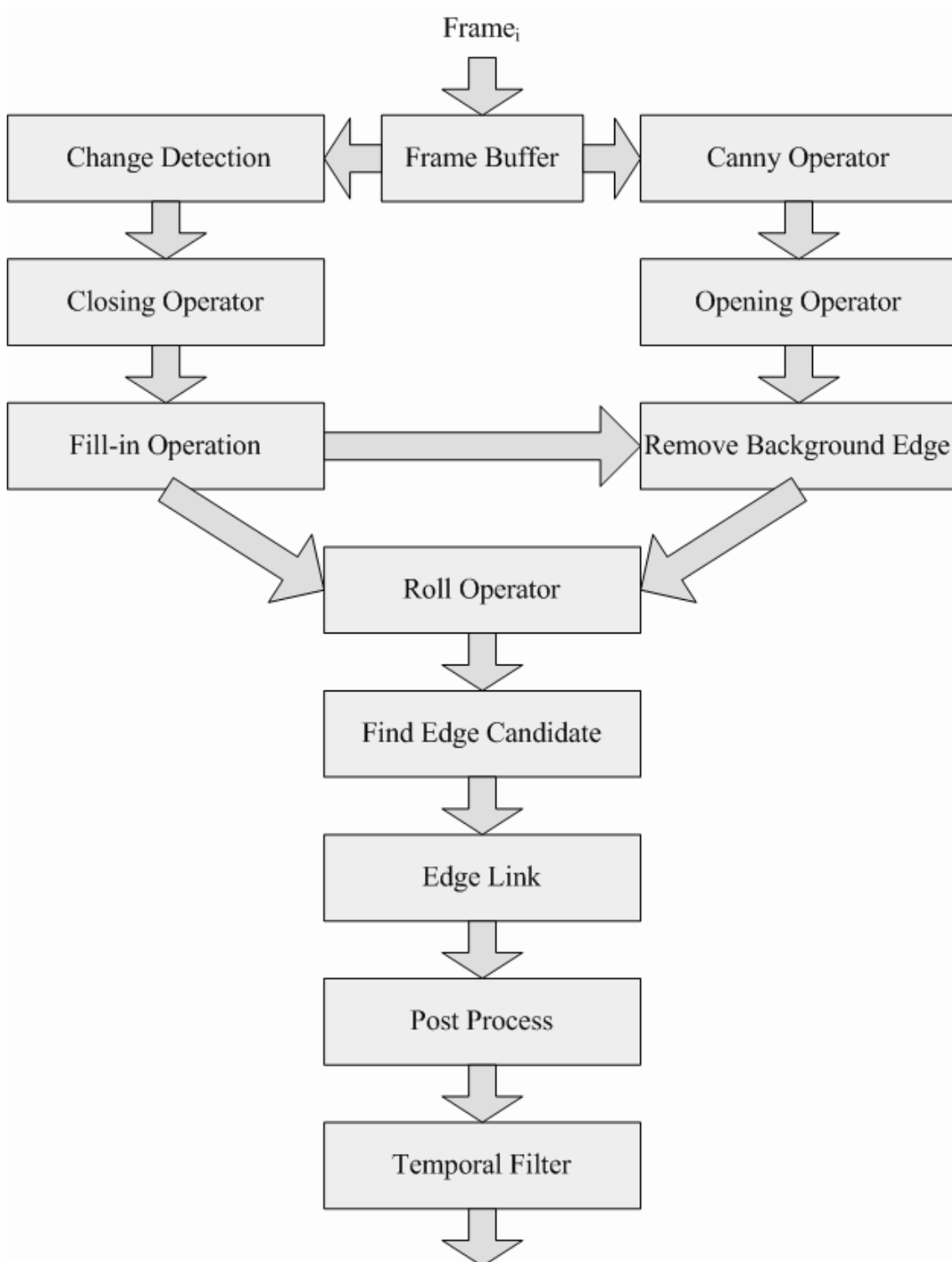


Figure 3.1: The block diagram of proposed method.

noise and therefore the stationary background usually has some difference between successive frames. In general, larger camera noise makes good segmentation more difficult to achieve. For example, when change detection-based technique is applied, the frame difference map of larger noise sequence (Figure 3.2) includes more background pixels than smaller noise sequence (Figure 3.3). It is apparent that the larger noise sequence needs more processing to obtain a better object mask.

3.2.2 Motivation for Noise Estimation

In the following steps of the system, we need many thresholds or parameters to make decisions and those thresholds are usually adjusted to counteract the influence of noise. We can adjust those parameter empirically but it is inconvenient since we have to tune them for different situations and it usually needs some experiences. In order to reduce the complexity of threshold decision, those parameters in the following steps are adjusted based on estimated camera noise.

3.2.3 Camera Noise Model

In the thesis, we assume the difference d_k of stationary pixels between successive frames obeys a zero mean Gaussian distribution $N(0, \sigma)$ with variance σ^2 , that is,

$$p(d_k|H_0) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{d_k^2}{2\sigma^2}\right\}. \quad (3.1)$$

where H_0 denotes the null hypothesis. As in [3], since the camera noise is uncorrelated between different frames, the variance σ^2 is equal to twice the variance of the assumed Gaussian camera noise distribution.

3.2.4 Procedure for Two-Stage Noise Estimation

In order to estimate the variance σ^2 , the sample space should include those pixels belonging to stationary background and exclude pixels belonging to moving objects. Our idea to discriminate the two kind of pixels is based on the observation which can be seen in Figure 3.4. The lighter pixels which represent larger difference are usually lumped together



Figure 3.2: A thresholded frame difference map of the Mother-and-Daughter sequence (from [8]).

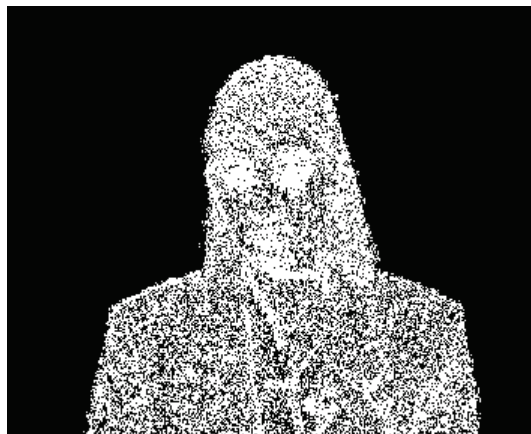


Figure 3.3: A thresholded frame difference map of the Akiyo sequence (from [8]).

or are distributed like a strip when they are introduced by moving objects. On the other hand, the larger frame differences caused by camera noise are usually randomly distributed. Hence, we reject the pixels whose neighbors have larger frame difference from the sample space during noise estimation.

We use the same mask of [11] to find out those pixels that belong to moving objects. For each pixel, we consider the four directional sums in frame difference map as shown in Figure 3.5. If one of the four directional sums is larger than certain threshold, we can assume that the pixel belongs to a moving object.

The following problem is how we choose the threshold. Up to the present, we can only calculate variance σ_G^2 of frame difference of entire frame, and therefore it is natural that we initially adjust the threshold based on σ_G^2 . If one of the four directional sums of a pixel is larger than $\alpha\sigma_G^2$, the pixel is classified to pixels influenced by moving objects. After we remove those pixels influenced by objects, those remaining pixels are used to estimate σ^2 . In order to verify the result of our method, we first manually choose the pixels belonging to stationary background to estimate the variance σ^2 . In Figures 3.6 and 3.7, the white areas are chosen to estimate σ^2 and the estimation result is regarded as exact. As we can see in Figures 3.8 and 3.9, this method can effectively remove most pixels influenced by moving objects.

It is obvious that the results in Figures 3.8 and 3.9 are still influenced by moving objects, because we adjust the threshold based on variance σ_G^2 of entire frame which has high relationship with moving objects. In order to reduce this problem, we use a two-stage noise estimation in this system. In the first stage, we use the $\alpha\sigma_G^2$ (In the experiment, α is 25) to be the threshold and get the variance σ_1^2 of stage one. In the second stage, we use the $\beta\sigma_1^2$ (In the experiment, β is 20) as the threshold and then we can obtain the final result σ_2^2 of stage two. The final result is shown in Figures 3.10 and 3.11. It can be seen that the result of two stage method is closer to exact value.

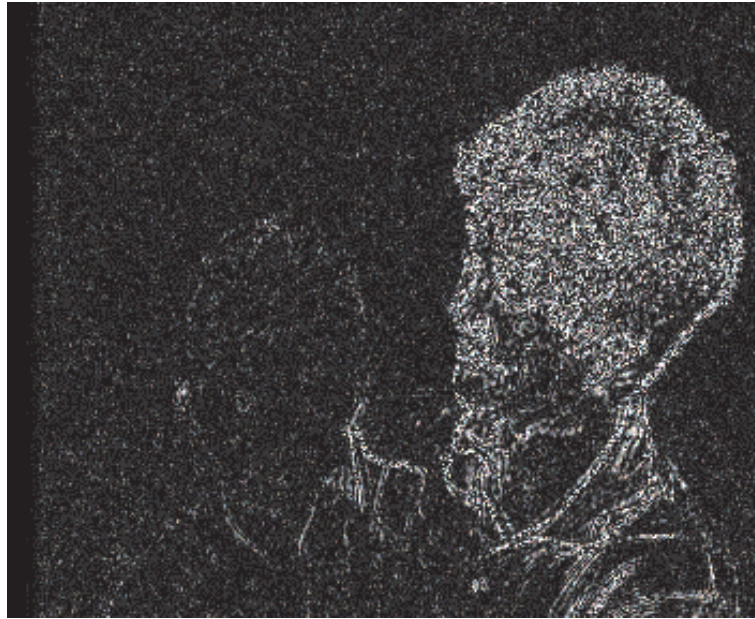


Figure 3.4: A frame difference map of Mother-and-Daughter sequence (from [8]).

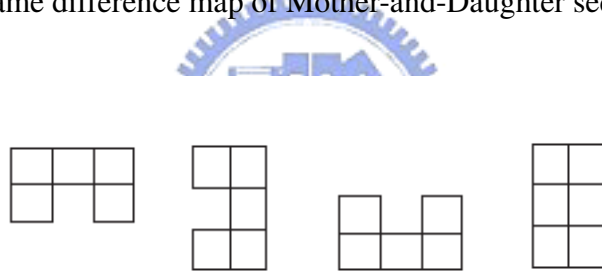


Figure 3.5: Four masks of directional sums (from [8]).



Figure 3.6: Mother-and-Daughter sequence (from [8]).



Figure 3.7: Claire sequence (from [8]).

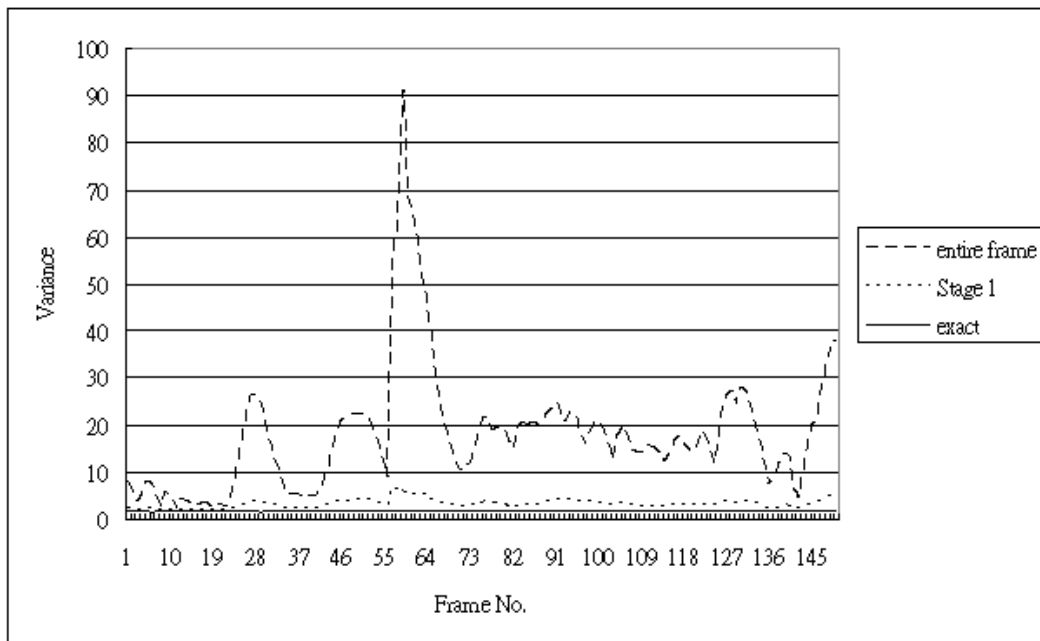


Figure 3.8: Noise estimation of Mother-and-Daughter sequence (from [8]).

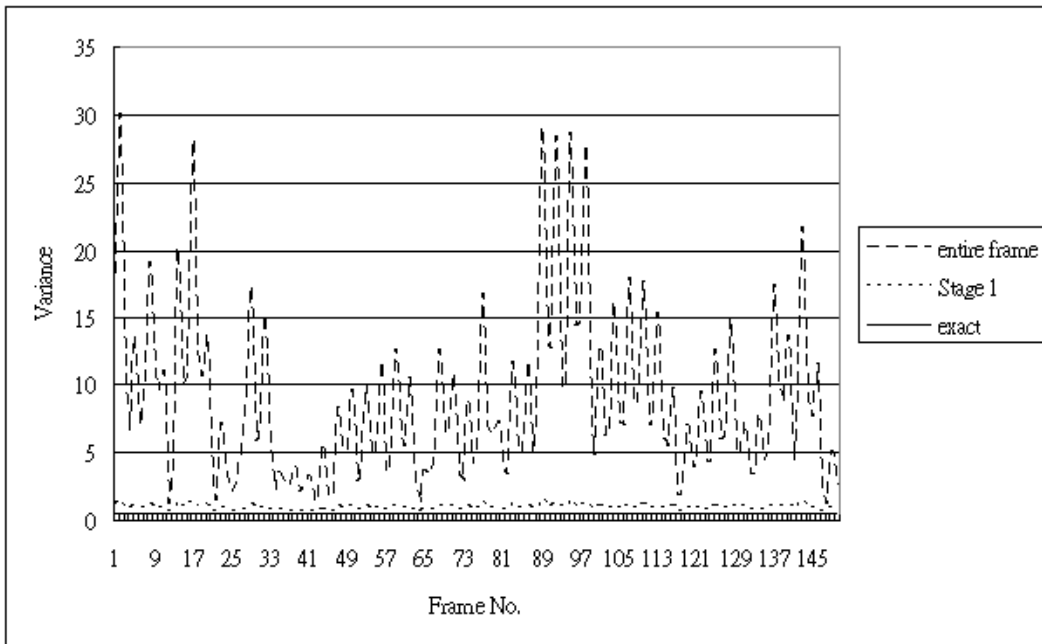


Figure 3.9: Noise estimation of Claire sequence (from [8]).

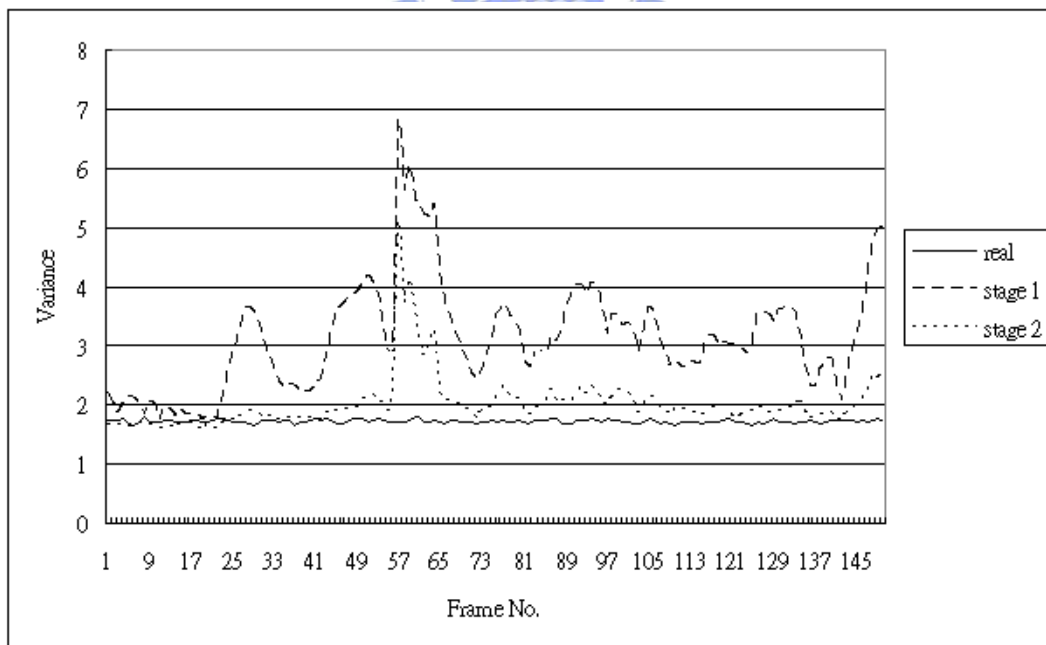


Figure 3.10: Noise estimation of Mother-and-Daughter sequence for the two-stage method (from [8]).

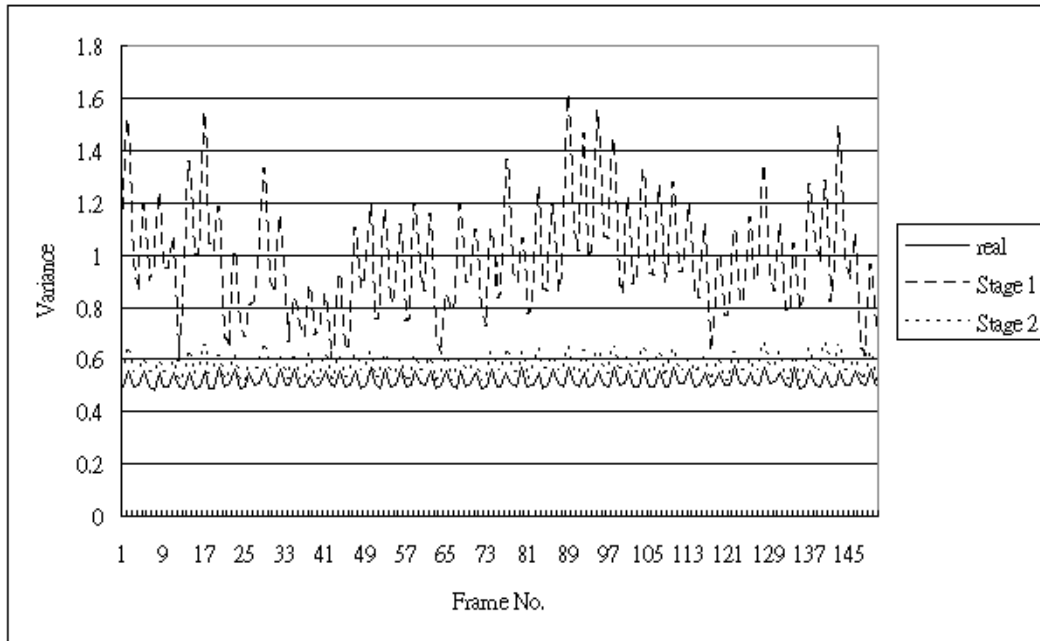


Figure 3.11: Noise estimation of Claire sequence for the two-stage method (from [8]).

3.3 Temporary Moving Object Mask

In this section, we will generate a temporary foreground mask and then the mask is used to get the outmost edge map of moving object. In this stage, we use change detection-based technique to obtain a rough mask. The major advantage of this technique is that the frame difference can be easily and speedily obtained. Then we use the binary morphological operator to remove small regions that arise due to camera noise. After these steps we can get a rough moving object mask in this stage.

3.3.1 Foreground Estimation

A moving object is also called foreground and the stationary parts are called background. At first, we use the 3×3 window to calculate the mean of squared frame difference for each pixel. If the result is larger than threshold, the pixel is classified as in a moving object. On the other hand, a pixel is classified as background when the result is smaller than threshold. The threshold here is adjusted based on the camera noise, that is, $\gamma\sigma^2$. The range of γ is from 20 to 100. An example of thresholded frame difference map is

shown in Figure 3.12.

3.3.2 Mathematical Morphological Operator

The output of thresholded frame difference map is binary image. The pixels of foreground are marked in white. As shown in Figure 3.12, there are many small regions which are marked in white and they do not belong to foreground. So we use morphology operator to delete them and make the pixels of foreground to lump together.

Mathematical morphology is a well-known set theoretic and shape oriented approach which treats the image as a set and the kernel of operation, dilation, erosion, opening, closing, etc., as another set. Each operation set, namely structuring element, plays an important role in dealing with shape or extracting features from the image. The structuring element is a kind of coefficients set in mathematical morphology. In digital image, there are two popular kinds of structuring elements, 4-connectivity and 8-connectivity.

In our application, we select the 3×3 and 8-connectivity structuring elements. The operation of binary erosion means if one of 8-connectivity pixels is marked black, then the central pixel is marked black. The operation of binary dilation means if one of 8-connectivity pixels is marked white, then the central pixel is marked white. The opening and closing operators are composed of erosion and dilation. In [9], closing operator can remove the salt noise and opening operator can remove pepper noise. So we employ closing operator to the thresholded frame difference map and the result is as shown in Figure 3.13.

3.3.3 Background Estimation

To obtain the robust moving object mask, we employ the background estimation in the same time.

The simplest way to judge whether the value of a pixel is background is to check the frame difference at this location. Since moving objects will cause larger frame difference, we can assume that a pixel belongs to background at this location when the frame difference at this location is very small from start to finish. For real-time application, it is



Figure 3.12: Threshold frame difference map of frame 27 in Claire sequence.



Figure 3.13: Threshold frame difference map of frame 27 in Claire sequence after closing operation.

impossible to make a decision after whole data is collected from beginning to end. In the system, we regard the value of pixel as background when its frame difference is small for some consecutive frames.

We consider six consecutive frames $f_k(i)$ ($1 \leq k \leq 6$) as the time of observation and a 3×3 window is used to calculate frame difference $d_m(i) = f_6(i) - f_m(i)$ ($1 \leq m \leq 5$) for each location i in a frame. For every location i , we calculate the mean and variance of $d_m(i)$ ($1 \leq m \leq 5$). If the variance is smaller than threshold, it means the changes between the six frames are small and we can regard the value of pixel at location i of sixth frame as background. The threshold here is also based on camera noise, that is, $\lambda\sigma^2$. The range of λ is from 2 to 10.

3.3.4 Obtaining Initial Object Mask

After foreground and background estimation, we can get two thresholded frame difference maps. In this stage, we want to generate the initial moving object mask. At first, we obtain a rough object by ORing the two maps. There might be noise present, which can be suppressed based on the size of connected components in the object mask. In the second step, we label the connected components in the object mask and set a threshold. In the experiment, the threshold is from 10 to 40. If the size of a component exceeds a threshold, we assume it belong to object mask. In the third step, we use the fill-in technique proposed in [5]. The fill-in technique is described as that at first we assign the pixels between the first and last white points of 3.18 to white points for each row. This procedure is then repeated for each column. The step-by-step results are shown in Figures 3.15 to 3.19.

3.3.5 Refining Initial Object Mask

Since the fill-in technique always marks the region between left and right boundaries, there are some pixels belong to background always regarded as objects. So we refine the object mask. It will be very helpful if the mask here is more accurate.

In this stage, we use the edge information to correct the initial mask and the Canny operator proposed in [1] is adopted to get edge information. The operator performs a gra-



Figure 3.14: The original frame 60 in Claire sequence.



Figure 3.15: Threshold frame difference map after foreground estimation of frame 60 in Claire sequence.



Figure 3.16: Threshold frame difference map after background estimation of frame 60 in Claire sequence.



Figure 3.17: The OR map of Fig 3.15 and Fig 3.16.



Figure 3.18: Threshold frame difference map after remove small region of frame 60 in Claire sequence.



Figure 3.19: Threshold frame difference map after fill-in of frame 60 in Claire sequence .

gradient operation on the image which convolutes it with Gaussian filter and then nonmaximum suppression is applied to thin the edge. In the last step, the thresholding operation with hysteresis is used to find and link edges. The thresholding operation including two thresholds: high-threshold and low-threshold. Pixels whose gradient is larger than high-threshold are regarded as edges and pixels whose gradient is smaller than low-threshold are regarded as non-edges. Pixels whose gradient is between high-threshold and low-threshold need to check their neighbors. If one of its neighbors is regarded as edge, these pixels are classified to edge. The edge map after applying Canny operator is shown in Figure 3.20. The related code of Canny operator is obtained from [12].

At first, we employ the opening operator to lump the pixel which is marked in black. The edge map after applying opening operator is shown in Figure 3.21. The way to refine the initial object mask is shrinking the initial mask to fit the edge map. The initial mask, edge map and shrunk mask are shown in Figures 3.22, 3.23 and 3.24, respectively, for Claire sequence. In these figures, we can see that the edge map includes many background edges and those background edges usually interfere with the final result. According to the initial moving object mask, when a position of object mask is marked black, we assume that there is a background edge in the position. The result after removing the background edge can be seen in Figure 3.22. In Figure 3.22, we can find some pixels which belong to background edges. To reduce the influence of those background edges, we use the connected algorithm to delete them. In Figures 3.23 and 3.24, we can see that background edges can be effectively removed and the final object masks are obtained.

3.4 Edge Linking for Moving Object

In our application, we employ the change detection technique to obtain a rough outline of an object. Next, we employ the Canny method [1] for edge detection and remove the background edges. Then, we employ roll operator [8] to shrink the initial object mask to edge map. Edge information is very useful in identifying object boundaries. However, typical edge detection algorithms, including the Canny method, do not necessarily yield closed region contours or fully connected curves at object boundaries. And the change



Figure 3.20: Edge map of frame 60 in Claire sequence.



Figure 3.21: Edge map after opening operator of frame 60 in Claire sequence.



Figure 3.22: Edge map after removing background edges of frame 60 in Claire sequence.



Figure 3.23: Final object edge map of frame 60 in Claire sequence.



Figure 3.24: Final object mask of frame 60 in Claire sequence.

detection mask is influenced by camera noise and the intensity of light. So it would lose some edges of moving object. In the section, the outermost edges in the region are identified and linked using a shortest path algorithm.

3.4.1 Find the Candidate of Edge Map

In this stage, we use the Mother-and-Daughter sequence for instance. The edge map after applying Canny operator is shown in Figure 3.25. The edge map after removing background edges is shown in Figure 3.26. Given a set of edges in a region, one common way to obtain a rough outline of the object is by orthogonal scans. Each row that contains edge pixels is considered. The space between the leftmost and the rightmost such pixels is filled in. Likewise, for each column that contains edge pixels, the space between the topmost and the bottommost such pixels is fill in. Then a rough object mask is obtained by ANDing the two pixel map. After the orthogonal scans, we use the 8-connectivity method to get the outmost edges of objects.

The method of 8-connectivity is that if the pixel is marked black and one of its 8-connectivity neighbors is marked white, we consider the pixel belong to the outmost edge pixel of moving object. The result can be seen in Figure 3.27. The others of the edge map is marked gray, as shown in Figure 3.28.

3.4.2 Overview of Shortest Path Algorithm

In [2], the author provides solutions to two graph problems under the following assumptions. There are n vertices, there exists at least one path between any pair of vertices, and the paths have positive lengths. The first problem is to construct a tree of minimum total length between n vertices, and the second problem is to find the path of minimum total length between two given vertices. In here, we consider the second problem.

To find the shortest path from vertex S to vertex t , the algorithm does the following:

1. Set a variable $D[x]$, x is any vertex. $D[x] = 0$ when x is equal to S , and $D[x] = inf$ when x is not equal to S . Set $y = S$.



Figure 3.25: The edge map of frame 49 after Canny operation in Mother-and-Daughter sequence.



Figure 3.26: The edge map of frame 49 after removing background edges in Mother-and-Daughter sequence.



Figure 3.27: The outmost edge map of frame 49 in Mother-and-Daughter sequence.



Figure 3.28: The weighted edge map of frame 49 in Mother-and-Daughter sequence.

2. Assume $D[x] = \min(D[x], D[y] + a(y, x))$, $a(y, x)$ is the distance from y to x .
Let $y = x$, if x is not equal to S and x never be assigned to y . For example:
 $D[S] = 0, D[a] = 5, D[b] = 7, D[c] = 6$, the result is $y = a$.
3. If $y = t$, the shortest path from S to t is found. Otherwise repeat step 2.
4. Calculate the total number of vertices from S to t and record the number of them.

3.4.3 Edge-Link Method

In this stage, we introduce the edge-link method in our system. After previous step, we can get the outermost edge map and weighted edge map of moving objects. We divide the edge map into two parts. Edge-linking is applied to each part independently. At first, we assign an equivalent length d_0 to the pixels which are marked in Figure 3.28, assign an equivalent length d_1 to the pixels which are marked white, and assign an equivalent length inf to the pixels which are marked black. We let $d_0 = 1, d_1 = 10$, and $inf = 10000$ in the experiments. To control the computational complexity, we may limit the search area to a band around the mask's boundary. We adopt two search areas, 22×22 and 46×46 , in our system.

Then we find out all vertices and the number of segments. We employ the connected algorithm to calculate the number of segments. We find out the coordinates of start vertex and termination. According to the coordinates, we can divide the direction into first and second quadrants. Then we can transfer the coordinate of vertex to a number. According to the number of vertex, we can employ the shortest path algorithm to link them from bottom to top. The Figure 3.29 shows the flowchart of edge-link algorithm. The result after edge-link is shown in Figure 3.30

3.5 Post-Processing

In this section, we want to refine the object mask and to extract the moving object. After linking the edges of moving object, we get the closed contour. At First, the object mask is padded with the edge pixels. The result is shown as Figure 3.31. Then we employ

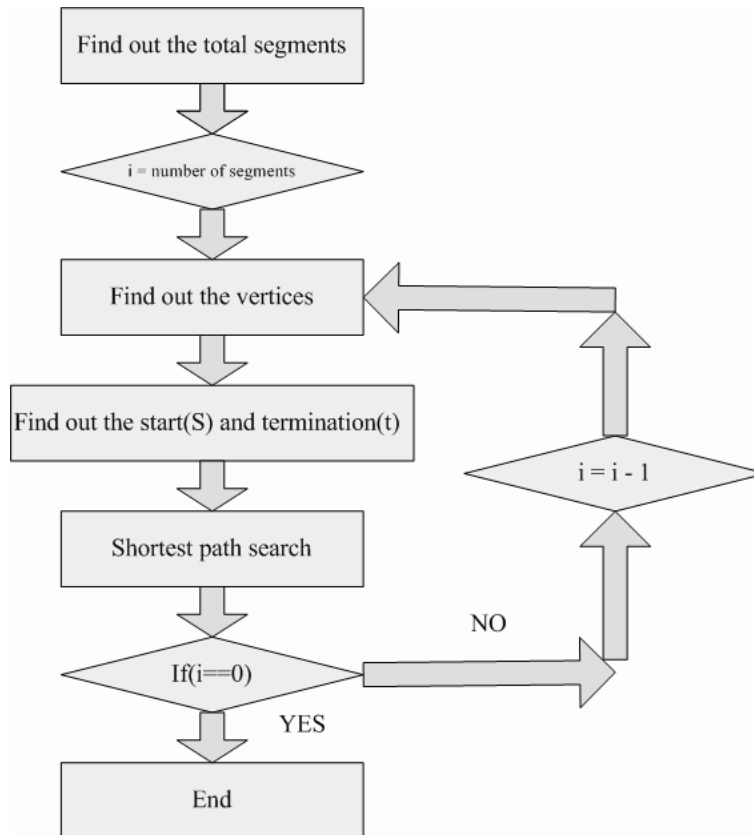


Figure 3.29: Flowchart of edge-link method.



Figure 3.30: The edge map of frame 49 after edge-link in Mother-and-Daughter sequence.

the connected algorithm to remove the redundant pixels of moving object and to extract moving object. The algorithm is based on the size of connected components in the object mask. We employ 8-connectivity method to label the connected components which is marked in white and set up a threshold. In the experiment, the threshold is 10000. If the size of a component does not exceed the value, we can assume it does not belong to the object mask. after the step we can get the accurate object mask. The original frame is shown in Figure 3.32. The result after post-processing is shown in Figure 3.33.

3.6 Temporal Filtering

Usually, the object mask does not change violently between the current frame and previous. Based on this, we employ the background registration technique. The flowchart is shown in Figure 3.34.

There is a buffer to store the object mask of previous frame in our system. When we get the first object mask, the temporal filter is started. The step-by-step procedure is as follow:

1. Weighting: Compare the object mask in previous frame and current frame. If the pixel at the same location between the previous and current frames is marked in different colors, we would mark the pixel in gray.
2. Registration: According to the weighting object mask, we prepare a buffer to record the accumulated value. If the pixel is marked black, we assume the pixel belongs to background and add a constant to the buffer. (In the experiment the constant is 10.) If the pixel is marked white, we know it belongs to foreground and we would set the buffer for zero. If the pixel is marked gray, we would subtract a constant to the buffer. (In the experiment the constant is 5.)
3. Object mask decision: We select a suitable threshold to judge the pixel belongs to background or not. (In the experiment the threshold is 50.) And we use the mask to extract moving object.



Figure 3.31: The object mask of frame 49 after padded with edge pixels in Mother-and-Daughter sequence.



Figure 3.32: The original frame 49 in Mother-and-Daughter sequence.



Figure 3.33: The moving object of frame 49 in Mother-and-Daughter sequence.

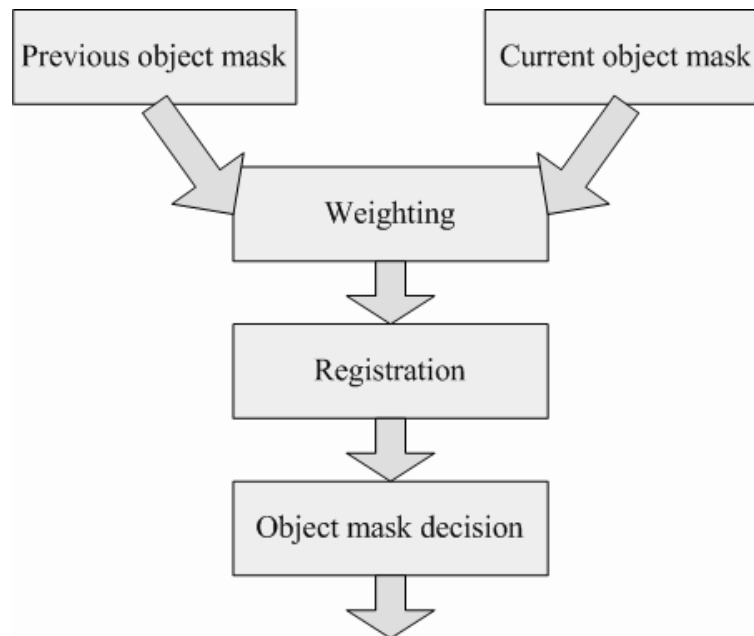


Figure 3.34: Flowchart of temporal filter.

The original frames of Claire sequence is shown in Figures 3.35(a), (b), and (c), respectively. The result without applying temporal is shown in Figures 3.35(d), (e), and (f), respectively. The result after applying temporal filter is shown in Figures 3.35(g), (h), and (i), respectively.

3.7 Conclusion

We summarize the proposed segmentation method. In the first step, the two-stage noise estimation is used to estimate the camera noise. As shown in section 3.2, this method can effectively remove the influence of moving objects. In the second step, we employ the change detection technique which includes foreground estimation and background estimation to get the initial object mask. To refine the initial object mask, we use mathematical morphological operators and connected algorithm to remove the undesirable region. For a better result, we use the edge information to refine the initial object mask. We get the edge map by applying the Canny operator, which is a good method for edge detection. Then we use the initial object mask to remove the background edges and use the connected algorithm to refine the edge map. According to the refined edge map and the



Figure 3.35: Segmentation of the Claire sequence. (a) The original frame 29. (b) The original frame 30. (c) The original frame 31. (d) Frame 29 without temporal filter. (e) Frame 30 without temporal filter. (f) Frame 31 without temporal filter. (g) Frame 29 with temporal filter. (h) Frame 30 with temporal filter. (i) Frame 31 with temporal filter.

initial object mask, we can get refined object mask. To get a more accurate object mask, the core of proposed segmentation method, edge-link method, is applied. However, since the behavior and characteristics of the moving objects differ significantly, the quality of segmentation result depends strongly on background noise, object motion, and the contrast between the object and the background. We adopt the temporal filter to maintain the reliable and consistent object information.

After all steps described as above are executed, we can get an rather accurate object mask for the entire sequence.



Chapter 4

Optimized Implementation on Personal Computer

In this chapter, we discuss the optimization of our implementation of video segmentation system on the personal computer. The optimization techniques are categorized two types, code acceleration and algorithm. We also discuss the performance of the optimization. Finally, we would show the performance after optimization in Claire sequence with CIF (352×288) format.



4.1 Optimization of Algorithm

In this section, we introduce the algorithmic optimization. To help us to locate and remove software performance bottlenecks, we employ the Intel VTune Analyzer [13] to collect, analyze, and display the performance data from the system-wide level down to the source level. The VTune Analyzer provides multiple profiling technologies that enables optimization across multiple operating system platforms and development environments and support the latest Intel process. Our system is applied in PC which is Intel Pentium M 1.733GHz with 1024-MB RAM.

At first, we employ the VTune Analyzer to collect the performance data of the original system as shown in Table4.1. We test the sequences Claire in our system.

In Table4.1, we can observe that over 50% of run time is consumed by the function

Table 4.1: Profile and Run Time Comparison of Claire Sequence (CIF)

Callee function	contribution (%)	Average run time (ms/frame)
labeling	52.3	776
edgelinek	14	208
TmpBack	9.6	142
Post_process	5.7	85
canny	4.796	70
erosion	2.5	37
dilation	2.5	37
NoiseEstimation	1.8	27
Roll	1.1	16
Another function	5.8	85
Total	100	1483

named *labeling()*. The function is to apply the connected algorithm. Recall from the last chapter that the connected algorithm is used frequently. So we first focus on the optimization of the function. To reduce the run time of the function and to maintain the accurate object mask, we review the flowchart of our system. According to the straight thinking we decrease the edge times of the *labeling()* function. It can obtain more speed-up.

Although we decrease the edge times of *labeling()* but it still consumes more computation. According to our experience, we find out the run-time of *labeling()* is correlated with the size of moving object. So we down-sample the initial object mask before applying the function of *labeling()*. The object mask after applying down-sample is shown in Figure 4.1. When the function is finished we up-sample the mask. The object mask after applying up-sample is shown in Figure 4.2. It can also obtain some speed-up.

We compare the run time of the original algorithm and the revised algorithm in Table 4.2.

4.2 Overview of Intel's MMX Technology

In previous section, we focus on the optimization of algorithm. Then we would focus on the Intel's multimedia extensions (MMX) instruction. We revise the code effectively and use the multimedia extensions (MMX) instruction to speed up.

4.2.1 The MMX Architecture [16], [17], [18]

The multimedia extensions (MMX) for the Intel Architecture (IA) were designed to enhance performance of advanced media and communication applications. The MMX tech-

Table 4.2: Run Time of Optimization of Labeling Function per Frame

Function	Original run time (ms)	Run time with optimization (ms)	Speedup (%)
Labeling	776	206	276.7



Figure 4.1: The object mask after applying down-sample.

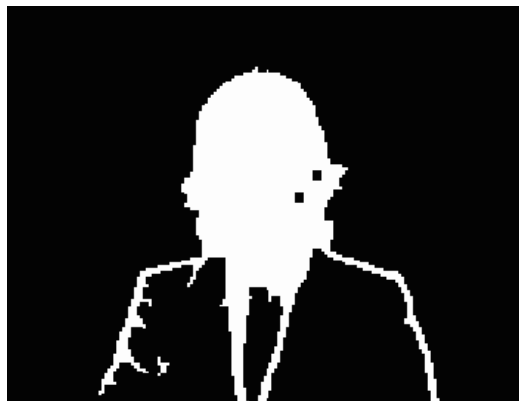


Figure 4.2: The object mask after applying labeling and up-sample.

nology introduces new general-purpose instructions. These instructions operate in parallel on multiple data elements packed into 64-bit quantities. These instructions accelerate the performance of applications with compute-intensive algorithms that perform localized, recurring operations on small native data.

The MMX technology uses the single instruction, multiple data (SIMD) technique. This technique speeds up software performance by processing multiple data elements in parallel, using a single instruction. The MMX technology supports parallel operations on byte, word, and doubleword data elements, and the new quadword (64-bit) integer data type.

The MMX technology defines a simple and flexible SIMD execution model to handle 64-bit packed integer data. This model adds the following new features to the IA: New data types, MMX registers and enhanced instruction set.

MMX Data Types

The MMX technology introduced the following four new 64-bit data types as illustrated in Fig. 4.3:



- Packed byte: 8 bytes packed into one 64-bits quantity.
- Packed word: 4 words packed into one 64-bits quantity.
- Packed doubleword: 2 doubleword packed into one 64-bits quantity.
- Packed quadword: One 64-bits quantity.

The 64 bits are numbered 0 through 63. Bit 0 is the least significant bit (LSB), and bit 63 is the most significant bit (MSB). The low-order bits are the lower part of the data element and the high-order bits are the upper part of the data element. Bytes in a multi-byte format have consecutive memory addresses. The ordering is little endian. That is, the bytes with lower addresses are less significant than the bytes with higher addresses.

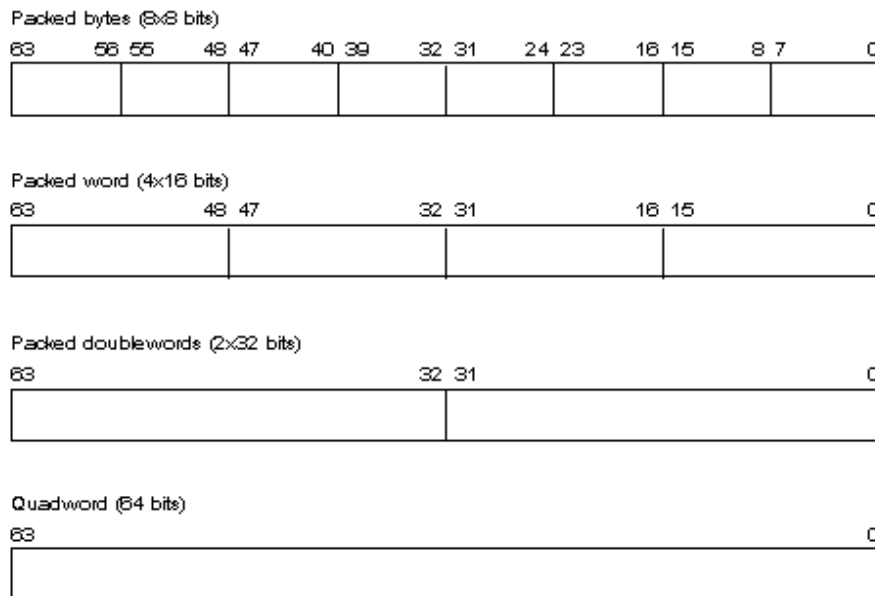


Figure 4.3: MMX packed data types (from [16]).

MMX Registers

The MMX register set consists of eight 64-bit registers as shown in Fig. 4.4, which are used to perform calculations on the MMX packed data but cannot be used to address memory. Values in MMX registers have the same format as a 64-bit quantity in memory. These registers are aliased to the floating-point registers. The MMX instructions access the MMX registers directly using the register names MM0 to MM7.

4.2.2 The MMX Instruction Set [18]

The MMX instructions are grouped into the following categories:

- Data transfer
- Arithmetic
- Comparison
- Conversion
- Unpacking

0	64
MM7	
MM6	
MM5	
MM4	
MM3	
MM2	
MM1	
MM0	

Figure 4.4: MMX register set (from [19]).

- Logical
- Shift
- Empty MMX state instruction (EMMS)

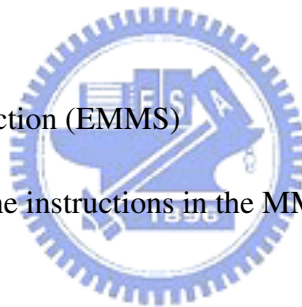


Table 4.3 gives a summary of the instructions in the MMX instruction set.

Data Transfer Instructions

We can transfer 32-bit or 64-bit data from memory to MMX registers and visa versa, or from integer registers to MMX registers and visa versa by a single instruction. We can transfer 32-bit data by MOVD and 64-bit data by MOVQ.

Arithmetic

The arithmetic instructions perform addition, subtraction, multiplication, and multiply-add operation on packed data types. For example, PADDB, PADDSB and PADDUSB instructions add signed or unsigned packed byte integers in wraparound mode, signed packed byte integers in signed saturation mode, unsigned packed byte integers in unsigned saturation mode, respectively.

Table 4.3: MMX Instruction Set Summary (from [19])

Category	Wraparound	Signed Saturation	Usinged Saturation
	32-bit Transfers		64-bit Transfers
Data Transfer			
Register to Register	MOVD		MOVQ
Load from Memory	MOVD		MOVQ
Store to Memory	MOVD		MOVQ
Arithmetic			
Addition	PADDB, PADDW, PADDQ	PADDSB, PADDSSW	PADDUSB, PADDUSW
Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSSW	PSUBUSB, PSUBUSW
Multiplication	PMULL, PMULH		
Multiply and Add	PMADD		
Comparison			
Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		
Conversion			
Pack		PACKSSWB, PACKSSDW	PACKUSWB

Table 4.4: MMX Instruction Set Summary (from [19])

Category	Wraparound	Signed Saturation	Unsigned Saturation
Unpack	32-bit Transfers		64-bit Transfers
Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		
Logical	Packed		Full 64-bit
And			PAND
And Not			PANDN
Or			POR
Exclusive OR			PXOR
Shift			
Shift Left Logical	PSLLW, PSLLD		PSLLQ
Shift Right Logical	PSRLW, PSRLD		PSRLQ
Shift Right Arithmetic	PSRAW, PSRAD		
Empty MMX State	EMMX		

Comparison Instructions

The comparison instructions compare the packed data in the source and destination operands for equal to or greater than. These instructions generate a mask of ones or zeros which are written to the destination operand.

Conversion Instructions

The conversion instructions perform conversions between the packed data types. For example, PACKSSDW instruction converts packed signed doubleword integers into packed signed word integers, using saturation to handle overflow conditions as shown in Fig. 4.5 for an example of the packing operation.

Unpack Instructions

The unpack instructions unpack bytes, words, or doublewords from the high- or low-order elements of the source and destination operands and interleave them in destination operand. By placing all 0s in the source operand, these instruction can be used to convert byte integers to word integers, word integers to doubleword integers, or doubleword integers to quadword integers.

Logical Instructions

The logical instructions perform bitwise logical operations on 64-bit quantities. For example, we can generate a zero register in MM0 by using “PXOR mm0, mm0.”

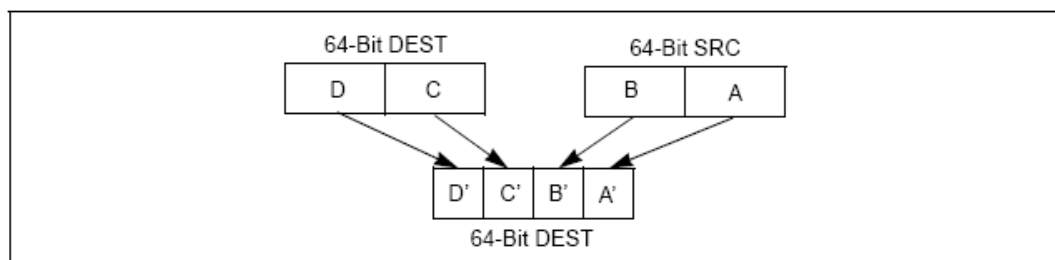


Figure 4.5: PACKSSDW instruction operation using 64-bit operands (from [18]).

Shift Instructions

The shift instructions have two types: logical shift and arithmetic shift. Logical shift instructions perform a logical left or right shift of the data elements and fill the empty high or low order bit position with zeros. Arithmetic shift instructions perform an arithmetic right shift, copying the sign bit for each data elements into empty bit positions on the upper end of each data elements.

EMMS Instructions

The EMMS instruction empties the MMX state. This instruction must be used to clear the MMX state at the end of an MMX routine before calling other routines that can execute floating-point instructions.

4.3 Code Acceleration

In this section, we focus on the code acceleration. In the original code we try to achieve the goal of video segmentation accurately. Now we want to revise the code effectively and do not change the result. According to VTune analyzer, we can find out where the function is more time-consumed. And we focus on them to speed up our system.

4.3.1 Labeling Function Optimization

Although we reduce the edge times of *labeling()*. According to VTune analyzer, we can find out that the *labeling()* is still more time-consumed. In this stage, we would optimize the code with MMX instruction. The code section is shown in Figure 4.6. The function is a subroutine of function *labeling()*. It deals with the size of connected region. When we set the labels to the connected region, then we would calculate the size of pixels of each label of connected region. After optimization, we get the modified function in Figure 4.7.

First, we move the data of *image_label* and *value* to register mm1 and mm2, respectively. The labels of eight continued pixels are stored in *image_label*. The label that we want to calculate the size of pixels is stored in *value*. In the same way, we define all

```

int calc_size(unsigned char *image_label,int label){
.....
for(i=0;i<frame_size;i++)
    if(image_label[i] == label) total++;
.....
}

```

Figure 4.6: Section of code that we would like to optimize with MMX.



```

DWORD64 one = 0x0101010101010101;
for(uv1=0;uv1<frame_size;uv1+=8)
{
    __asm{
        pusha
        mov eax,image_label
        mov ebx,value
        mov ecx,counter
        add eax,uv1
        movq mm1,[eax]
        movq mm2,[ebx]
        movq mm4,one
        pcmpeqb mm1,mm2
        pand mm4,mm1
        movq [ecx],mm4
        popa
        emms
    }
    total += (counter[0] + counter[1] + counter[2] + counter[3]+
             counter[4] + counter[5] + counter[6] + counter[7]);
}

```

Figure 4.7: Code in Fig. 4.6 after optimization using MMX instructions.

the bytes of the register mm4 1 and view the register mm4 as the criterion with 1. Next, we could compare mm1 with mm2 and set the byte of mm1 to 1 if the byte of mm1 and mm2 is equal. Then we apply the AND operation to mm4 and mm1 and set the byte of mm4. In this step, we could imply the function “ $(image_label[i] == label)$ ”. Finally we accumulate the value of counter and it could be implied as function “ $total + +$ ”.

After optimization, we compare the run time of the original code with the revised code in Table 4.5.

4.3.2 Edgeline Function Optimization

In this stage, we focus on the function *edgeline()*. The function *edgeline()* is to connect the segments of boundary of moving object.

1. Replace *malloc()* function by *calloc()* function:

According to the VTune analyzer, we find out the function *malloc()* which is provided with C library is more time-consuming. So we replace it by function *calloc()*. The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by size and whose value is unspecified. The *calloc()* function shall allocate unused space for an array of *n* elements each of whose size in bytes is *esize*. The space shall be initialized to all bits 0. The original code with *malloc()* function is shown in Figure 4.8, and the revised code with *calloc()* function is shown in Figure 4.9. After this procedure, it can reduce 50 milliseconds time-consumed.

2. Add the *break* instruction to the section of original code:

Table 4.5: Run Time of Optimization of Labeling Function per Frame

Function	Original run time (ms)	Run time with optimization (ms)	Speedup (%)
Labeling	206	188	9.57

```

typedef struct{
Vertex_Type vexs[MaxVertexNum];
Edge_Type edges[MaxVertexNum][MaxVertexNum];
int n,e; } MGraph;

G= (MGraph *) malloc(sizeof(MGraph));

```

Figure 4.8: The section of original code with malloc function.

```

Vertex_Type *vexs;
vexs = (Vertex_Type *) calloc (MaxVertexNum,sizeof(Vertex_Type));
Edge_Type edges[MaxVertexNum][MaxVertexNum];
int n,e;

```

Figure 4.9: The section of revised code with calloc function.

In our idea, we want to find out the vertex from down to up. In the original code, it find out the all vertices and the coordinate of the last vertex is what we want. In the revised code it search the vertex from down to up and when we find out the vertex it would leave the loop. The original and revised codes are shown in Figure 4.10 and Figure 4.11, respectively. After this procedure, it can reduce 28 milliseconds time-consumed.

```

for(i=0;i<y;i++)
for(j=0;j<x;j++){
if(sign[j+i*x]==1 ){
rs = i ;
cs = j;      }
}

```

Figure 4.10: The section of original code without break instruction.

```

for(i=y-1;i>0;i--){
  int flag=0;
  for(j=x-1;j>0;j--){
    if(sign[j+i*x]==1){ rs = i ;
                        cs = j ;
                        flag=1;  }
    if(flag==1) break;
  }
  if(flag==1) break;
}

```

Figure 4.11: The section of revised code with break instruction.

After optimization, we compare the run time of the original code and the revised code in Table 4.6.

4.3.3 Post_processing Function Optimization

We use MMX instruction for optimization. The code section is shown in Figure 4.12. The function is to pad the edge pixels to object mask. After optimization, we get the modified function in Figure 4.13. The algorithm is introduced as follow. First, we make all the bytes in register mm0 0. Next, we move data of *image_mask* and *edge* to mm2 and mm3 respectively. Then we could compare mm0 with mm2 and set the byte of mm0 is 255 if the byte of mm2 is 0. Finally, the data of mm3 could subtract the mm0 and store the result in register mm3. After optimization, we compare the run time of the original code with the revised code in Table 4.7.

Table 4.6: Run Time of Optimization of Edgelinek Function per Frame

Function	Original run time (ms)	Run time with optimization (ms)	Speedup (%)
edgelinek	208	130	60

```

for(uv1=0;uv1<frame_size;uv1++)
  if(edge[uv1] == 0) image_mask[uv1] = 0;

```

Figure 4.12: Section of code that we would like to optimize with MMX.



```

for(uv1=0;uv1<frame_size;uv1+=8)
{
  __asm{
    pusha
    mov eax,edge
    mov ebx,image_mask
    add eax,uv1
    add ebx,uv1
    pxor mm0,mm0 // let mm0 be zero

    movq mm2,[eax]
    movq mm3,[ebx]
    pcmpeqb mm0,mm2
    psubusb mm3,mm0
    movq [ebx],mm3
    popa
    emms      }
}

```

Figure 4.13: Code in Fig. 4.12 after optimization using MMX instructions.

Table 4.7: Run Time of Optimization of Post_process Function per Frame

Function	Original run time (ms)	Run time with optimization (ms)	Speedup (%)
post_process	85	68	25

4.3.4 Erosion and Dilation Function Optimization

According to VTune analyzer, we focus on *erosion()* and *dilation()* to speed up our system. In our application, we select the 8-connectivity structuring element. The operation of binary erosion means if one of 8-connectivity neighbor pixels is marked black, the central pixel is marked black. The section of original code is shown in Figure 4.14. When one of 8-connectivity neighbor pixels is marked black, the central pixel is marked black and we can ignore the statements below. So we add the *continue* instruction for optimization. After optimization, we get the modified function in Figure 4.15. It is the same way to binary dilation operator for optimization. The run time after optimization is shown in Table 4.8.

Table 4.8: Run Time of Optimization of Erosion and Dilation Function per Frame

Function	Original run time (ms)	Run time with optimization (ms)	Speedup (%)
erosion	37	22	68.18
dilation	37	22	68.18


```

for(i=1; i<(height-1); i++)
  for(j=1; j<(width-1); j++){
    int index=j+i*width;
    if(image_in3[j-1+(i-1)*width ] == 0) image_out3[index] = 0;
    if(image_in3[j+(i-1)*width  ] == 0) image_out3[index] = 0;
    if(image_in3[j+1+(i-1)*width] == 0) image_out3[index] = 0;
    if(image_in3[j-1+i*width     ] == 0) image_out3[index] = 0;
    if(image_in3[j+1+i*width     ] == 0) image_out3[index] = 0;
    if(image_in3[j-1+(i+1)*width ] == 0) image_out3[index] = 0;
    if(image_in3[j+(i+1)*width  ] == 0) image_out3[index] = 0;
    if(image_in3[j+1+(i+1)*width] == 0) image_out3[index] = 0; }

```

Figure 4.14: The section of original code without continue instruction.



```

for(i=1; i<(height-1); i++)
  for(j=1; j<(width-1); j++){
    int index=j+i*width;
    if(image_in3[j-1+(i-1)*width ] == 0) { image_out3[index] = 0; continue; }
    if(image_in3[j+(i-1)*width  ] == 0) { image_out3[index] = 0; continue; }
    if(image_in3[j+1+(i-1)*width] == 0) { image_out3[index] = 0; continue; }
    if(image_in3[j-1+i*width     ] == 0) { image_out3[index] = 0; continue; }
    if(image_in3[j+1+i*width     ] == 0) { image_out3[index] = 0; continue; }
    if(image_in3[j-1+(i+1)*width ] == 0) { image_out3[index] = 0; continue; }
    if(image_in3[j+(i+1)*width  ] == 0) { image_out3[index] = 0; continue; }
    if(image_in3[j+1+(i+1)*width] == 0) { image_out3[index] = 0; continue; } }

```

Figure 4.15: The section of original code with continue instruction.

4.4 Conclusion

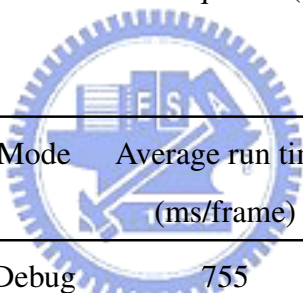
In the other functions, we also use the similar optimization with MMX instruction. After optimization, we compare the run time of the original system with optimization of Claire sequence. The result is shown in Table 4.9. The run time per frame for *labeling()* is reduced from 776 milliseconds to 188 milliseconds in debug mode. The run time per frame for *edgeline()* is reduced from 208 milliseconds to 130 milliseconds in debug mode. The run time per frame for *erosion()* is reduced from 37 milliseconds to 22 milliseconds in debug mode. The run time per frame for *post_process()* is reduced from 85 milliseconds to 68 milliseconds in debug mode. The run time per frame for entire system is reduced from 1483 milliseconds to 755 milliseconds in debug mode. The speed-up of entire video segmentation system is 96.42%. The comparison of run time between debug and release mode is shown in Table 4.10.



Table 4.9: Profile and Run Time Comparison of Claire Sequence (CIF) in Debug Mode

Callee function	contribution (%)	After opt. (%)	Average run time (ms/frame)	After opt. (ms/frame)
labeling	52.3	24.9	776	188
edgelinek	14	17.2	208	130
TmpBack	9.6	15.4	142	116
Post_process	5.7	9.2	85	68
canny	4.7	9.6	70	73
erosion	2.5	2.8	37	22
dilation	2.5	2.8	37	22
scale	-	3.1	-	24
NoiseEstimation	1.8	3.4	27	26
Roll	1.1	2.3	16	18
Another function	5.8	9.3	85	68
Total	100	100	1483	755

Table 4.10: Run Time Comparison of Claire Sequence (CIF) in Debug Mode and Release Mode



Mode	Average run time (ms/frame)
Debug	755
Release	261

Chapter 5

Simulation Results

In the last chapter, we discussed how we optimized some functions of the proposed system. In this chapter, we present some simulation results and run-time analysis in our system.

5.1 Segmented Object Masks



In this section, we show some simulation results of Mother-and-Daughter, Claire, Akiyo, and LabI sequences with CIF format. The Claire and Akiyo sequences are commonly test sequence in videoconferencing, Mother-and-Daughter is the case which include two major objects, and LabI is a natural video sequence which is captured in our lab. It can be seen that we can get more accurate object masks in the proposed system.

5.1.1 Segmentation Results

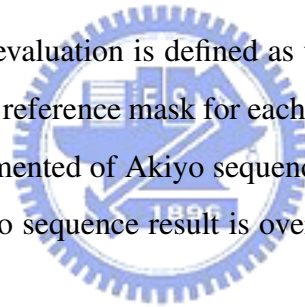
Some results of Mother-and-Daughter sequence, Clare sequence, Akiyo sequence, and LabI sequence in Figures 5.1– 5.4, respectively.

As we can see, the boundary of mask is very accurate. There are two major factors to influence the segmentation result. First, the boundary of moving object depends on the camera noise. In the case of low camera noise sequence, we can set more critical thresholds in foreground estimation and background estimation and get more accurate object

boundary. The camera noises of the four sequences from high to low are Mother-and-Daughter, LabI, Claire and Akiyo. According to the camera noise, the accuracy of object mask is in inverse order. If the camera noise is more great, the global threshold value would not get accurate object mask in entire sequence. For example, we only get accurate object mask in several frames of Mother-and-Daughter sequence because its huge camera noise. Second, if the object in the sequence moves fast, the boundary of object would cover background. Because these covered background become uncovered, the related boundary is usually incorrect, such as in Figure 5.4. To overcome the question, we find that if the background is flat and without other things although the object moves fast but we still can eliminate the covered background in the system.

5.1.2 Performance Evaluation

The criterion for performance evaluation is defined as the fractional agreement between our segmentation result and the reference mask for each frame which is proposed in [20]. We take MPEG-4 released segmented of Akiyo sequence comparing with our simulation result. The percentage of Akiyo sequence result is over 99% for all frames. We can see the result in Figure 5.5.



5.2 Run-time Analysis

In this section, we would show the comparison of run time between the original and the optimized system in Akiyo and LabI sequences with debug mode and we would show the run time with different format such as CIF (352×288) and QCIF (176×144) formats. The profiles of Akiyo and LabI sequences are shown as Tables 5.1 and 5.2. If the camera noise is large and cannot obtain accurate boundary of moving object, it would consume more computation in edge-link function. Table 5.3 shows the run time and frame rate of different sequences (CIF or QCIF format) in release mode in Visual C++. When we process the sequence with QCIF format in release mode, we can achieve the requirement of real time application. The fast frame rate would be over 20 frames per second.

As we can see, the run time of sequence with CIF format is not fast enough. So we

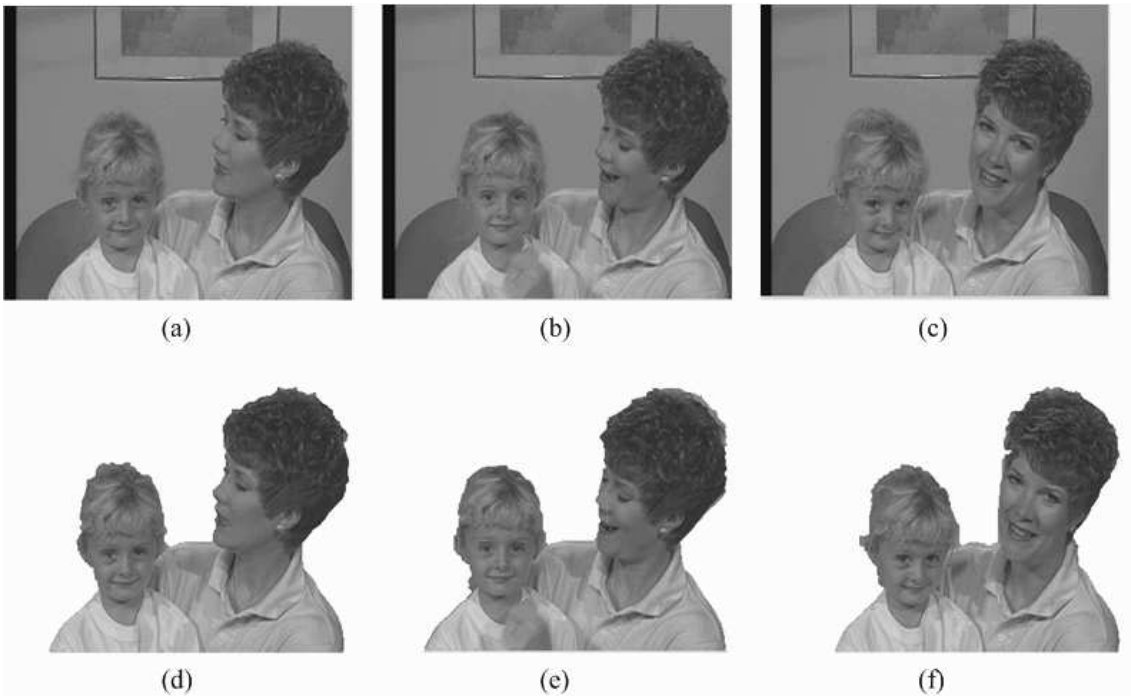


Figure 5.1: Segmentation of the Mother-and-Daughter sequence. (a) Frame 48. (b) Frame 156. (c) Frame 234. (d) Segmented object in frame 48. (e) Segmented object in frame 156. (f) Segmented object in frame 234.

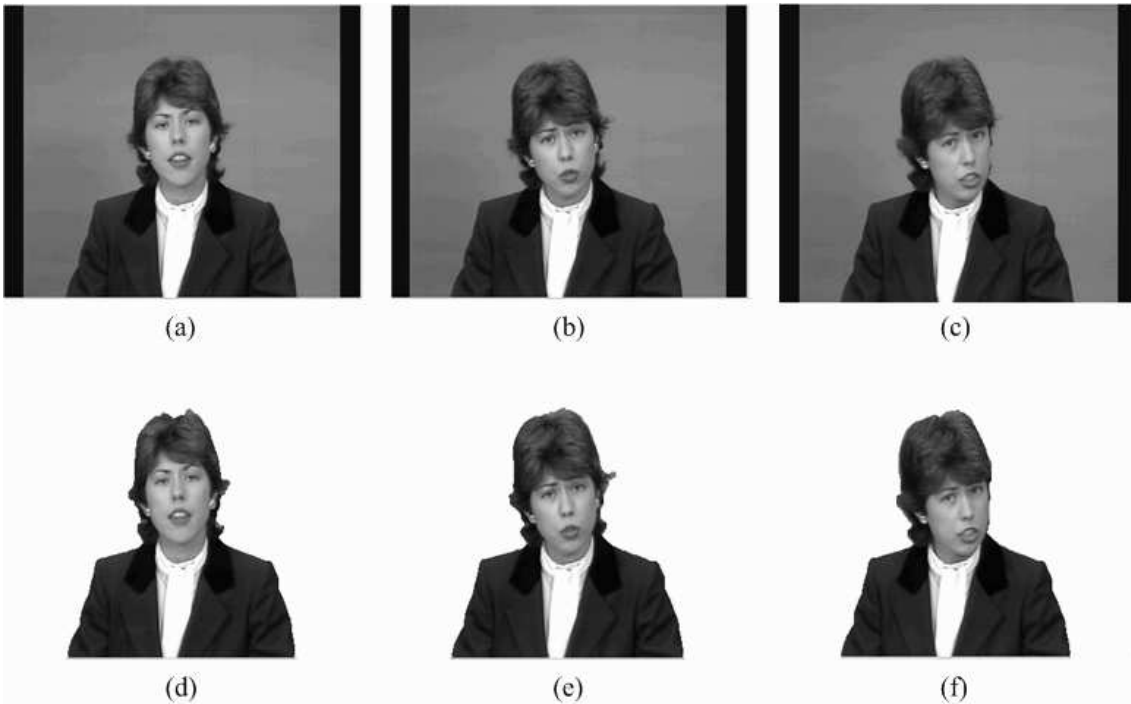


Figure 5.2: Segmentation of the Claire sequence. (a) Frame 16. (b) Frame 63. (c) Frame 104. (d) Segmented object in Frame 16. (e) Segmented object in Frame 63. (f) Segmented object in Frame 104.

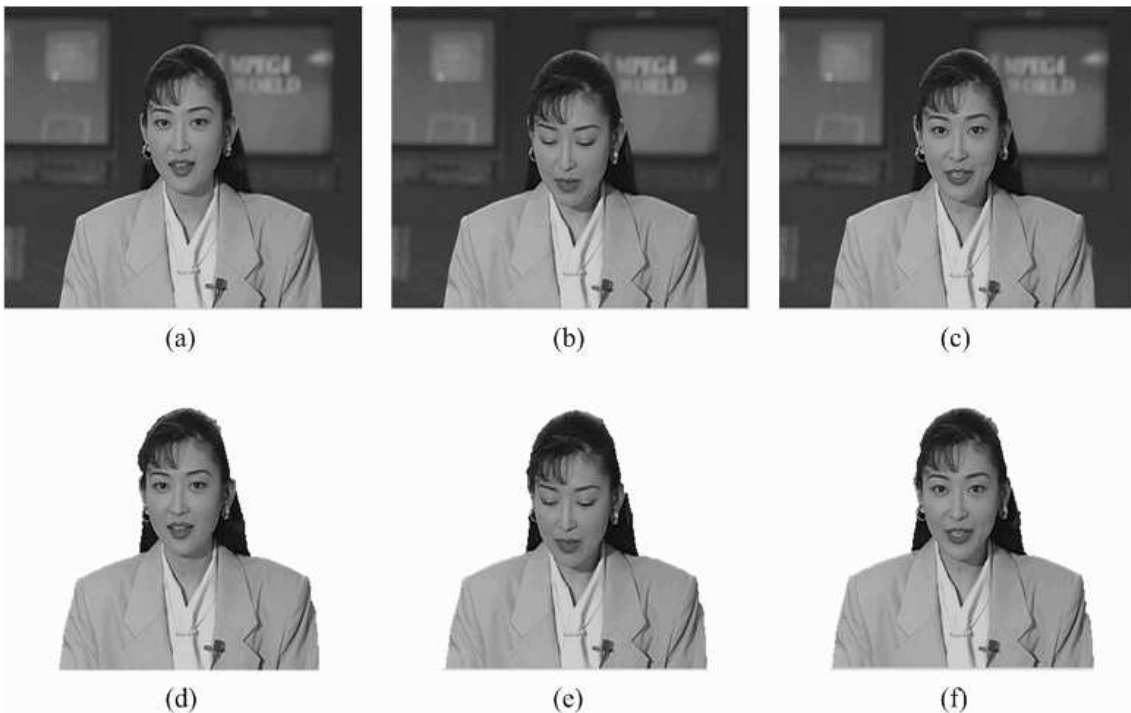


Figure 5.3: Segmentation of the Akiyo sequence. (a) Frame 24. (b) Frame 60. (c) Frame 141. (d) Segmented object in Frame 24. (e) Segmented object in Frame 60. (f) Segmented object in Frame 141.

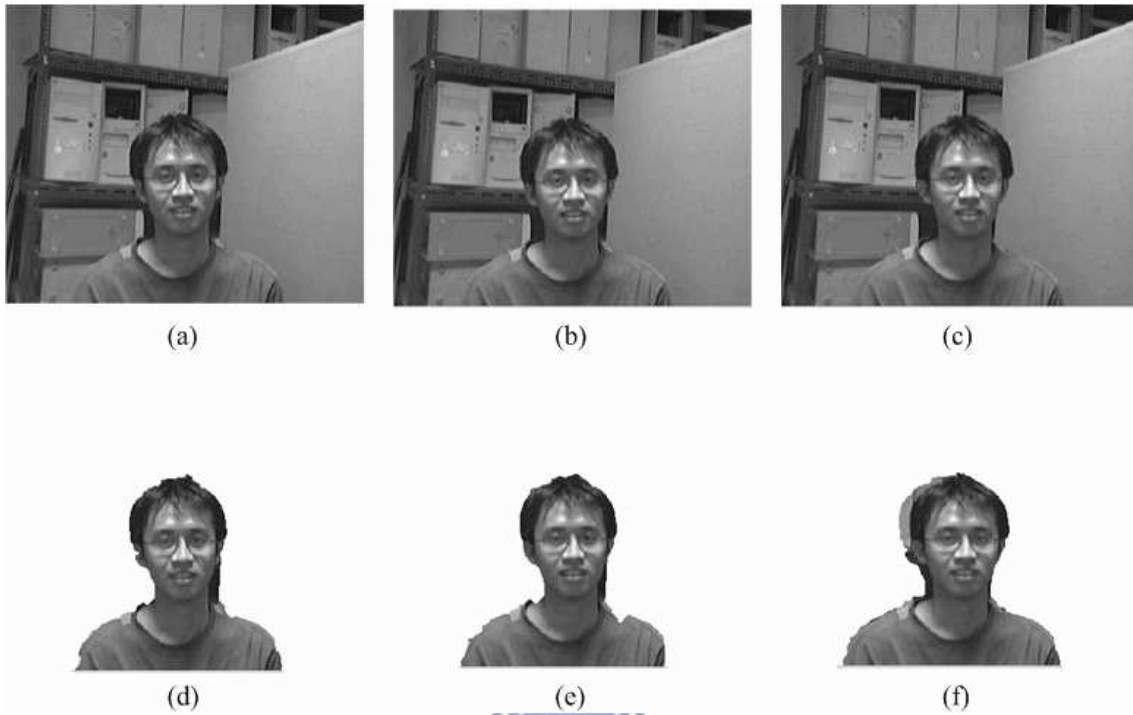


Figure 5.4: Segmentation of the LabI sequence. (a) Frame 18. (b) Frame 19. (c) Frame 20. (d) Segmented object in frame 18. (e) Segmented object in frame 19. (f) Segmented object in frame 20.

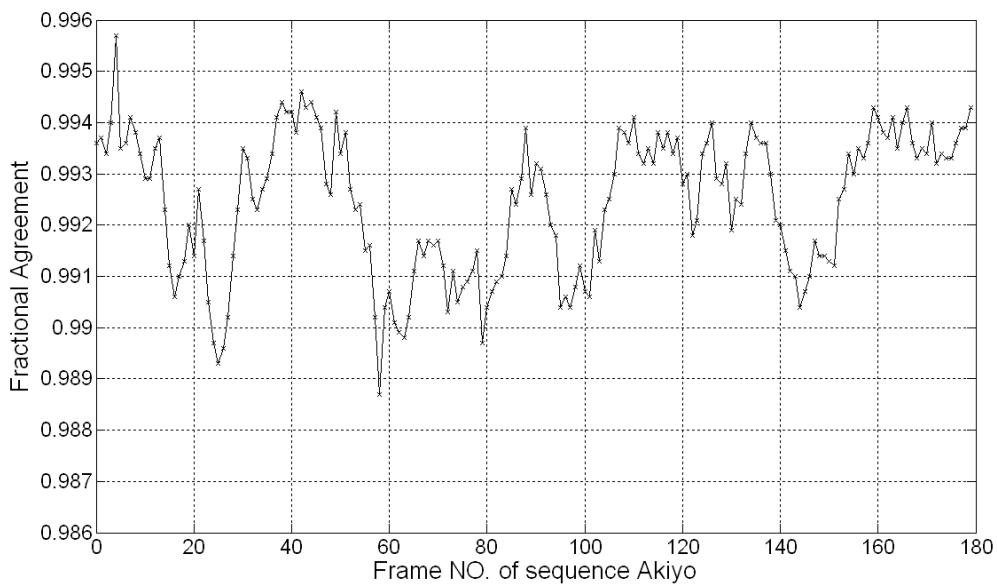


Figure 5.5: Fractional agreement between the segmented foreground object of the Akiyo sequence and the reference mask.

also propose the simple segmentation method which can speed up the system but it can not maintain the accuracy of segmentation result for the entire sequence. We remove the function of edge-link and background estimation in original segmentation system and the result is shown as in Table 5.4. The block diagram is shown in Figure 5.6. The frame rate could be 12 frames per second. We also compare our algorithm with [23] as shown in Table 5.5.

5.3 Complexity Analysis

In the following analysis, the designation “data” in front of a dash indicates that the operation is associated with data values(memory contents), whereas the designation “mem” indicates that operation is associated with memory address. The reason for distinguishing “data” and “mem” operations is that many processors treat these two types of operation differently.

We calculate the number of addition and multiplication in major algorithms. In Table 5.6, we estimate the necessary computations for completing each major algorithms of one CIF frame of Claire sequence. According to our observation, the complexity would be associated with the size of moving object. If the moving object is bigger, the complexity would increase.

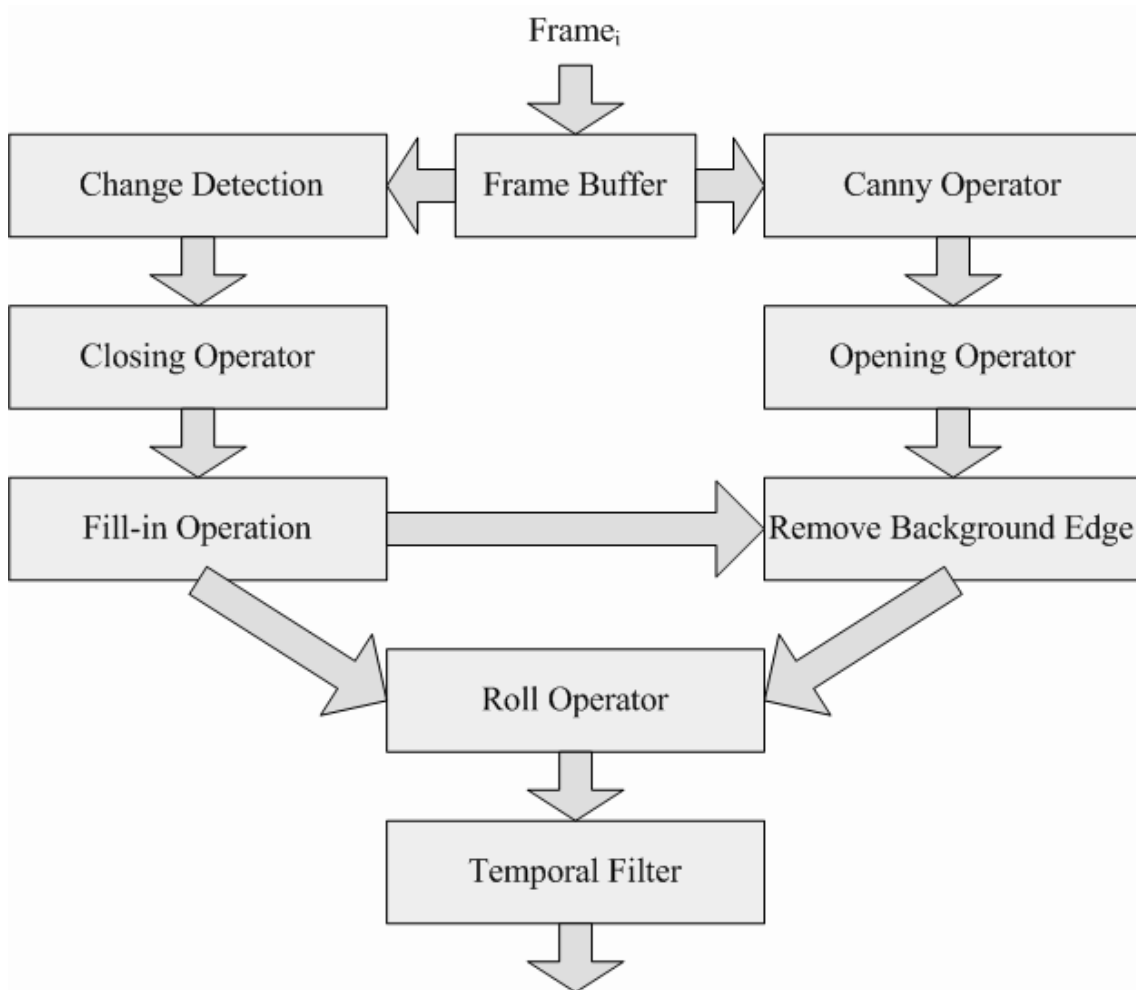


Figure 5.6: Flowchart of simple segmentation method.

Table 5.1: Profile and Run Time Comparison of Akiyo Sequence (CIF) in Debug Mode

Callee function	contribution (%)	After opt. (%)	Average run time (ms/frame)	After opt. (ms/frame)
labeling	29.5	21.6	413	164
edgelinek	30.2	17.2	422.8	130
TmpBack	13.3	19.2	186.2	146
Post_process	7.3	10.5	102.2	80
canny	5.2	9.4	72.8	71
erosion	2.1	2.2	29.4	17
dilation	2.2	2.4	30.8	18
scale	-	3.1	-	23
NoiseEstimation	2	3.2	28	24
Roll	1.1	2	15.4	15
Another function	7.1	9.2	99.4	70
Total	100	100	1400	758

Table 5.2: Profile and Run Time Comparison of LabI Sequence (CIF) in Debug Mode

Callee function	contribution (%)	After opt. (%)	Average run time (ms/frame)	After opt. (ms/frame)
labeling	26.3	15.3	526	160
edgelinek	37.4	41.2	748	431
TmpBack	7.3	11.2	146	117
Post_process	4.9	7.3	98	76
canny	3.9	7.7	78	81
erosion	6.4	2.1	128	22
dilation	5.1	1.9	102	20
scale	-	2.3	-	24
NoiseEstimation	1.4	2.5	28	26
Roll	1.1	2.1	22	22
Another function	6.2	6.4	124	66
Total	100	100	2000	1045

Table 5.3: Run Time and Frame Rate of Different Sequences with CIF or QCIF Formats in Release Mode

Sequence (format)	Average run time (ms/frame)	Frame rate (frame/sec)
Claire (CIF)	261	3.83
Akiyo (CIF)	284	3.52
LabI (CIF)	364	2.75
Claire (QCIF)	49	20.41
Akiyo (QCIF)	66	15.16

Table 5.4: Run Time and Frame Rate of Different Sequences use Simple Segmentation Method in Release Mode

Sequence (CIF)	Average run time (ms/frame)	Frame rate (frame/sec)
Claire	106	9.43
Akiyo	86	11.63
LabI	100	10

Table 5.5: Comparison of Run-time between Different Algorithms

Algorithm	Processor (CPU)	Test sequence (QCIF)	Average run-time (ms/frame)	Frame rate (frame/sec)
Proposed	Pentium M 1.733-GHz	Claire	49	20.41
Proposed	Pentium M 1.733-GHz	Akiyo	66	15.16
Predictive Watershed [23]	Pentium III 800-MHz	Not mentioned	106.64	9.4

Table 5.6: Complexity of Major Algorithms in One CIF Frame of Claire Sequence

Algorithm (function name)	Mem-add	Mem-mult	Data-add	Data-mult	Data-comparison
Connected algorithm					
(labeling)	1,915,163	613,549	56	0	2,336,959
Edgeline algorithm					
(edgeline)	26,762,567	24,725,864	702,408	128,026	46,959,042
Background Estimation					
(TmpBack)	11,081,708	0	23,270,247	1,611,792	8,260,434
Canny Operation					
(Canny)	6,835,508	1,512,960	3,923,490	2,301,566	2,056,984
Erosion					
(erosion)	258,087	85,817	100,100	100,100	85,817
Dilation					
(dilation)	51,510	16,958	100,100	100,100	16,958
Roll Operation					
(Roll)	6,760,576	4,260,179	0	0	198,115

Chapter 6

Integration of System on Personal Computer

In this chapter, we demonstrate the implementation on the personal computer. The system block diagram is shown in Figure 6.1.

The overall architecture of our segmentation system is shown in Figure 6.2. Here, we need a digital camera to capture images, a personal computer to control the system and to segment the image, and a displayer to show the final result.

We adopt the GUI (Graphed User Interface) mode and it is implemented by Windows SDK (Software Development Kit) which is developed by Microsoft for high-level computer languages to easily implementation.

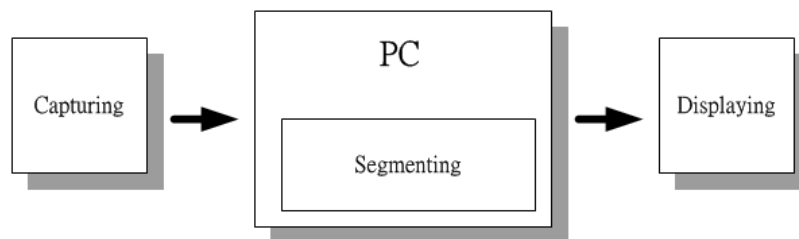


Figure 6.1: System block diagram (from [8]).

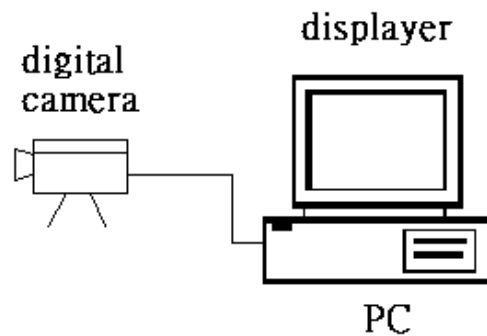


Figure 6.2: Circumstance of implementation (from [8]).

6.1 Video Capturing

6.1.1 Video for Windows

In this system, the input image is captured by digital camera. To control the operation of capturing, a standard video capturing method, named VfW (abbreviation of Video for Windows), in the Microsoft OS is adopted. Video for Windows version 1.0 was released in November 1992 for the Windows 3.1 operating system and was optimized for capturing movies to disk [14]. This SDK provides applications with a simple, message-based interface to access video and waveform-audio acquisition hardware and to control the process of streaming video capture to disk. Besides, VfW helps with connectivity to device driver and retrieve the capability and information of it.

6.1.2 AVI Format

In VfW, AVI is the mostly used format. The captured raw frame is embedded in an AVI file which can be extracted for segmentation input.

AVI stands for Audio-Video Interleaved. Figure 6.3 shows the hierarchical structure. Refer to header file (vfw.h in Visual C++) for complete information about parsing AVI file. To extract video data, we use a file parser that simply locate `##db` and copy suitable length of data following that.

RIFF	RIFF universal file header
AVI	AVI file header
Hdr1	Header list
Avih	AVI header
Str1	List of stream header for each stream in the AVI file
Strh	Video or audio stream header
Strf	Video or audio stream format
JUNK	Used to align data
##wb	Audio frame data
##vb	Video frame data

Figure 6.3: AVI header (from [8]).

6.1.3 Implementation of Capture

The implementation of capture is aided by a free application called AVICap from [15]. It contains three steps:

1. Create capture handle:

An AVICap capture window handles the details of streaming audio and video capture to AVI files and it provides a flexible interface for applications. The video capture can be add to application by the code shown in Figure 6.4.

2. Parameter modification:

After initializing driver window handler, some fundamental parameters should be confirmed to ensure captured data fit system requirement, such as the code shown in Figure 6.5.

3. Capture operation:

In this step, we start to capture image from digital camera and related code is shown in Figure 6.6. Here, the captured image which is AVI format is stored in a buffer

```
//Create the capture window
hwndC = capCreateCaptureWindow("Video Capture
Window",WS_CHILD | WS_VISIBLE,0,0,352,288,hwnd, 0);
// Connect the capture window to the driver
capDriverConnect(hwndC, 0);
// Get the capabilities of the capture driver
capDriverGetCaps(hwndC, &caps, sizeof(caps));
```

Figure 6.4: Related code for creating a capture window.

```

// Set the preview rate in milliseconds
capPreviewRate(hwndC,30);
// Start previewing the image from the camera
capPreview(hwndC, TRUE);

```

Figure 6.5: Related code for parameter modification.

and then the required video data is extracted from the buffer. Finally, the extracted data is sent to the module of video segmentation.

6.2 Result Display

After we finish the video segmentation, we need to display the result on the displayer. The procedure to create a display window is similar to previous section. After creating a window, we can show the result on displayer according to the related code in Figure 6.7.

6.3 Graphed User Interface

There are two major control unit: capture-control unit and threshold-adjustment unit. The capture-control unit controls every option needed for digital camera, such as start, stop, image size, and luminance. The threshold-adjustment unit is used to adjust the related threshold in foreground estimation and background estimation. The application program is shown in Figure 6.8. The window at left side is the preview captured image and the window at right side is the extracted moving object. The window below is the message

```

char filename[] = "c:\\buffer.avi" ;
unsigned char Y_Component[352*288*3/2];
capFileSetCaptureFile(hwndC,filename);
FptrIn = fopen(filename,"rb");
capCaptureSingleFrameOpen(hwndC);
capCaptureSingleFrame(hwndC);
capCaptureSingleFrameClose(hwndC);
fseek(FptrIn,0xa08,SEEK_SET);
fread(Y_Component,1,352*288*3/2,FptrIn);

```

Figure 6.6: Related code for capture operation.

```

hDC = GetDC(hwnd_5);
for(i2=0;i2<288;i2++)
{
    for(i3=0;i3<352;i3++)
    {
        int index = i2*352 + i3 ;
        int tmpY = image_regy[index] ;
        int tmpU = image_regu[index] - 128 ;
        int tmpV = image_regv[index] - 128 ;
        SetPixel(hDC,i3,i2,RGB(tmpR,tmpG,tmpB) );
    }
}
ReleaseDC(hwnd_5,hDC);

```

Figure 6.7: Related code for displaying.

window which shows the related threshold and the frame rate.

6.4 Conclusion

In the system, the image size is CIF (352×288) and the PC is Intel Pentium M 1.733GHz with 1024-MB RAM. We use the digital camera which is produced by company Logitech to capture image. For quickly obtaining an apparent improvement, we use the simplification segmentation method which is introduced in chapter 5. According to our observation, the GUI mode would consume more computation. It is because the function *SetPixel*. If there is no moving object existed, the frame rate of demonstration is 4 frames per second. Otherwise, the frame rate is 3 frames per second. If illumination of the video sequence is better and the background is flat, the moving object can be extracted accurately.



Figure 6.8: Application program interface.

Chapter 7

Conclusion and Future Work

We developed and implemented of an video segmentation system on the personal computer.

The core of our system was the graph-based edge linking technique. We used the change detection technique to get the object mask roughly. For easier obtaining the relative thresholds of each module a two staged method for camera noise estimation was introduced to reduce the effect of moving objects and those thresholds were adjusted based on the estimated camera noise. We used the Canny operation to detect the edge of entire frame and we removed the edges of background by the change detection mask and the object mask of previous frame. Then we shrunk the object mask to the edge map by roll operation and we could get the object mask. To refine the object mask, we employed edge-link method, postprocessing, and temporal filter which was based on the background registration technique.

If the format of video sequence was QCIF (176×144), the proposed segmentation method could achieve 20 frames per second. For CIF format application, we also proposed a simple segmentation method. The frame rate could achieve 12 frames per second. Simulation results showed that our algorithm could give correct segmentation results very quickly. According to these features, our algorithm was very suitable for videophone and videoconference with stationary background.

For quality improvement we can do some improvements for the main projects in the future.

1. Adding the module to deal with shadow and light change.

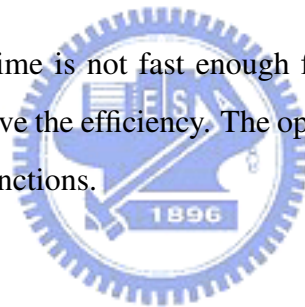
The position of shadow is controlled by the position of light and therefore the shadow effect greatly depends on the position of light. If the great shadows appear in background, the shadows are also regarded as moving objects in the module of background subtraction. Hence, a module to reduce the shadow effect can improve the accuracy of final image mask when the shadow is great.

2. Combine the segmentation system with MPEG-4 encoder.

The MPEG-4 Encoder is completed by another member in our laboratory. In the future, we will combine the segmentation system and MPEG-4 encoder to achieve a whole system.

3. More optimization.

The current processing time is not fast enough for CIF format and a better optimization is need to improve the efficiency. The optimization may focus on labeling, edgeline, and SetPixel functions.



Bibliography

- [1] J. F. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, pp. 679–698, Nov. 1986.
- [2] D. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, pp. 269–271, 1959.
- [3] T. Aach, A. Kaup, and R. Mester, "Statistical model-based change detection in moving video," *Signal Processing*, vol. 31, pp. 165–180, Mar. 1993.
- [4] S. Y. Ma, S. Y. Chien, and L. G. Chen, "Efficient moving object segmentation algorithm using background registration technique," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 7, pp. 577–586, July 2002.
- [5] T. Meier and K. N. Ngan, "Video segmentation for content-based coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 8, pp. 1190–1203, Dec. 1999.
- [6] E. P. Ong, B. J. Tye, W. S. Lin, and M. Etoh, "An efficient video object segmentation scheme," in *IEEE Int. Conf. Acoust. Speech Signal Processing*, vol. 4, 2002, pp. 3361–3364.
- [7] H. Luo and A. Eleftheriadis, "Rubberband: an improved graph search algorithm for interactive object segmentation," in *IEEE Int. Conf. Image Processing*, vol. 1, 2002, pp. 101–104.
- [8] Y-H. Lin, "Real-time video segmentation based on background modeling for video-conferencing," M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R. O. C., June 2004.

- [9] R. C. Gonzales and R. E. Woods, *Digital Image Processing*. Addison-Wesley, 1992.
- [10] C. Kim and J. N. Hwang, “Fast and automatic video object segmentation and tracking for content-based application,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 2, pp. 122–129, Feb. 2002.
- [11] Y-H. Jan and D. W. Lin, “Video segmentation with extraction of overlaid objects via multi-tier spatio-temporal analysis,” *Int. J. Electrical Engineering*, vol. 11, no. 3, pp. 205–218, Aug. 2004.
- [12] “Canny operator code,” <http://ouray.cudenver.edu/na0alber/DataCompressionPaper.htm>.
- [13] Intel, *Getting Started With the VTune (TM) Performance Analyzer*. 2003.
- [14] “Video for Windows,” http://msdn.microsoft.com/library/psdk/multimed/avifile_8dgz.htm.
- [15] “VidCap: full-featured video capture application,” <http://msdn.microsoft.com/library/devprods/vs6/visualc/vcsample/vcsmpvidcap.htm>.
- [16] Intel, *MMX Technology — Programmers Reference Manual* 2000.
- [17] Intel, *IA-32 Intel Architecture Software Developer’s Manual*, vol. 1. 2003.
- [18] Intel, *IA-32 Intel Architecture Software Developer’s Manual*, vol. 2. 2003.
- [19] M-Y. Liu, “Real-time implementation of MPEG-4 video encoder using SIMD-enhanced Intel Processor,” M.S. thesis, Degree Program of Electrical Engineering and Computer Science, National Chiao Tung University, Hsinchu, Taiwan, R. O. C., July 2004
- [20] M. Wollborn and R. Mech, “Refined procedure for objective evaluation of VOP generation algorithms,” Doc. ISO/IEC JTC1/SC29/WG11 MPEG98/3448, Mar. 1998
- [21] T. Meier and K. N. Ngan, “Automatic segmentation of moving objects for video object plane generation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, pp. 525–538, Sep. 1998.

- [22] E. Sifakis, L. Grinias, and G. Tziritas, "Video segmentation using fast marching and region growing algorithms," *EURASIP J. Applied Signal Processing*, vol. 8, pp. 379–388, Apr. 2002.
- [23] S. Y. Chien, Y. W. Huang, and L. G. Chen, "Predictive Watershed: A Fast Watershed algorithm for video segmentation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 5, pp. 453–461, May 2003.
- [24] W. Si, Z. Yong-dong, and L. Shou-xun, "An automatic segmentation algorithm for moving objects in video sequence under multi-constraints," in *IEEE Int. Conf. Multimedia Expo*, vol. 1, June 2004, pp. 555–558.
- [25] W. Wei, K. N. Ngan, and N. Habili, "Multiple feature clustering algorithm for automatic video segmentation," in *IEEE Int. Conf. Acoustics Speech and Signal Processing*, vol. 3, May 2004, pp. 625–628.
- [26] "Demo page of video segmentation," <http://video.ee.ntu.edu.tw/web/demo/seg.htm>.

