國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering
College of Electrical Engineering & Computer Science
National Taiwan University

Master Thesis

智慧型監控系統中物件追蹤平台之演算法及硬體架構設計

Algorithm and Hardware Architecture Design of Video Object Tracking Framework for Smart Surveillance Network

曾鈺翔
Yu-Hsiang Tseng

指導教授：簡韶逸 博士
Advisor: Shao-Yi Chien, Ph.D.

中華民國 98 年 10 月

October, 2009


國立臺灣大學
博碩士論文
學術倫理・誠實信用

Algorithm and Hardware Architecture Design of Video Object Tracking Framework for Smart Surveillance Network

By
Yu-Hsiang Tseng

THESIS

Submitted in partial fulfillment of the requirement
for the degree of Master of Science in Electronics Engineering
at National Taiwan University
Taipei, Taiwan, R.O.C.

Oct. 2009

Approved by:

Advised by:

Approved by Director :


國立臺灣大學
博碩士論文
學術倫理・誠實信用

國立臺灣大學（碩）博士學位論文
口試委員會審定書

智慧型監控系統中物件追蹤平台之演算法及硬體架構設計

Algorithm and Hardware Architecture Design of Video Object Tracking Framework for Smart Surveillance Network

本論文係曾鈺翔君（學號:R96943030）在國立臺灣大學電子工程學研究所完成之碩（博）士學位論文，於民國九十八年十月十五日承下列考試委員審查通過及口試及格，特此證明

口試委員：

簡韶逸

林嘉文（指導教授）　　[illegible]

劉志尉　　賴尚宏

系主任、所長


國立臺灣大學
博碩士論文
學術倫理・誠實信用

誌謝

經過兩年多的研究生生活，我的碩士生涯終於要畫下一個句點，這兩年多來，真的學習到很多有用的知識與經驗。這本論文得以出版，必須感謝許多人的幫忙與指導，在此，對長久以來給我支持與鼓勵的大家給予由衷的感謝。

首先我必須感謝的是從我大學四年級就開始指導我的指導教授　簡韶逸博士，他在我這一路上耐心地指點我、教導我，使我獲益良多。在教授三年多來的帶領下，我得以一窺這研究的世界，也由衷地佩服這位以身做則教導我們研究的好老師。再來，我必須感謝詹偉凱學長（凱哥），當我大學四年級時開始跟著老師做專題研究時，便是他在帶領著我做研究，並且指導我許多設計的技巧。並且，他也幫助我決定了研究的方向，也耐心地和我一同研究演算法及架構上所遇到的問題，也給了我在論文及口試上不少重要的建議。此外，則瑋學長及瑞欣學長（Larry）也曾給我不少在專業領域上的建議，真的很高興能有你們這些厲害的學長，也真的很感謝你們。

其次要感謝實驗室的學長和同學們。功炎是我一開始修課時重要的同組伙伴，也是我第一次參加晶片設計比賽時的同組戰友，他開朗大方的個性使我能夠很快地融入這個實驗室；志豪（豪哥）在修課的時候，給我不少的指導與幫忙，使我能在碩一裡拿下不錯的成績；翰儒、俊英（小白）、慶懿，都在工作站的使用上給予我不少的協助，讓我在剛當上實驗室工作管理員時能夠比較容易上手。再來是在 MD404 一起研究、修課、玩樂的同學們：東興（Paulman）真的是我們實驗室的領袖，只要有不懂的東西，問他就對了，不管是工作站管理員、同組修課等等，他都是一個非常可靠的伙伴；家恆（Lkince）在研究上真的給予我很大的協助，他畢業後在工作時，當我請教他問題時，他也耐心地指導，幫助我解決問題；遊戲達人誠豪（豪弟）及認真做事愛打橋牌的耿賢（Nana）真是我們實驗室的開心果，總是帶給我們歡笑，他們兩人良好的默契也常成為大家歡樂的源頭，「厲害」；熱愛台南的泳霖（Cary）很成功地向我們推薦台南的好，也當起地頭蛇帶我們深刻地認識台南這個好地方，並且，他也是在 Paulman 及 Lkince 畢業後我重要的飯咖；最後是閃亮亮的雅婷（Shimii），會「不小心」在實驗室放出陣陣的亮光，她也是我大學以來一個很好的朋友。這幾年有了你們，讓我的研究生生活更加美好，我也會很懷念這段日子的。

最後，我要感謝我的父母及姐姐，他們默默地支持我的選擇，在我遇到困難時，鼓勵著我、支持著我，雖然我總是很晚回家，但他們總是等著我的回來，桌上總是有著熱呼呼的湯溫暖著我的胃。還有，陪伴我 14 年卻在口試前一週永遠離開我的 Nicky，他會一直活在我的心中的。最後，我還必須感謝我的女朋友采薇，在我失落不順的時候，一直陪伴著我及不斷地想辦法使我開心起來，讓我能夠提起勇氣繼續奮鬥。

再次向所有人說一聲：謝謝。以此論文獻給所有關心及支持我的人。


國立臺灣大學
博碩士論文
學術倫理・誠實信用

中文摘要

近年來監控系統的使用大幅地成長，因此，監控系統中之智慧型功能也會變得越來越重要，例如物件追蹤、前景分割以及人臉偵測等等，否則我們必需要從冗長的監視影像中花費很多時間及精力來得到我們所想要的資訊。

而在追蹤的演算法中，我們發現物體在經過光影變化或是有著與物體相似色的地方時，追蹤的演算法容易出現問題。因此我們改進現有的粒子濾波器框架並設計出一個能夠處理這兩種問題的追蹤演算法。

以外，我們設計了一個協處理器來處理及時運算，並且能夠支援並加速包含我們所提出之追蹤演算法在內的一些常用在智慧型監控系統中的演算法。我們使用了子字平行、資料串流以及硬體共享的技術來解決現今多媒體處理器常會遇到的問題：高產量需求、高頻寬需求、可程式化以及低成本。在我們的硬體中，支援一些基本的圖型運算如直方圖累積、CORDIC 等等。而基於我們的硬體設計，可以使用我們的硬體來加速如物件追蹤、前景分割、人臉偵測、行為辨識等等的演算法。並且，這協處埋器是可重組化且可以用來當作是新演算法的測試平台。

原型晶片利用聯華電子 90nm 技術製成，面積為 3.26×3.26mm^2，其工作頻率為 125MHz，最大消耗功率為 33.3 mW。而此晶片最多可以在 640×480、每秒 30 張、YUV 4:2:0 的影像輸入中同時追蹤 10 個物件。並且，由於使用子字平行技術，我們可以省下約 60.28%的晶片記憶體。


國立臺灣大學
博碩士論文
學術倫理・誠實信用

Algorithm and Hardware Architecture Design of Video Object Tracking Framework for Smart Surveillance Network

Advisor: Shao-Yi Chien

Author: Yu-Hsiang Tseng

Graduate Institute of Electronics Engineering

National Taiwan University

Taipei, Taiwan, R.O.C.

October 26, 2009

Contents

List of Figures

List of Tables

Abstract

Surveillance systems are widely used today and the number of employed cameras increases. The intelligent functions in surveillance are more and more important for helping user inspecting the video content. Among the intelligent functions, tracking and segmentation are the most widely used. In this work, we try to address the problems that will happen when we want to track video objects under light condition changes and video objects with background-alike color. We proposes a enhanced particle filter framework that can handle these two kind of problems. Moreover, in order to achieve the real-time applications, we also design a hardware coprocessor that can support most of the operations used in intelligence surveillance system, including those operations used in our proposed algorithm. In order to overcome the typical hardware design challenge in accelerating vision algorithms, such as high throughput requirement, high bandwidth requirement, programmability and low cost, we employ sub-word level parallelism, streaming based processing, and hardware sharing techniques. In this design, basic image processing operations, such as histogram accumulation, CORDIC, window operations, are supported. Specific operations, such as those in video object segmentation, are also supported. Based on the hardware design, many applications, such as tracking, segmentation, face detection, feature detection and description, and motion analysis, can be accelerated. This coprocessor is reconfigurable and it can be used to test the new developed algorithms. We also implement the hardware coprocessor as a chip with standard cell based design flow. The prototype chip is fabricated with UMC 90nm technology. The chip size is $3.260 \times 3.260 mm^2$. The

external bandwidth is estimated, and the chip can support video object segmentation and, at the same time, tracking 10 objects in 640×480 30 fps 4:2:0 YUV color sequence with 125 MHz clock frequency and 33.3mW power consumption. Moreover, with sub-word level parallelism, the on chip memory saving is 60.28%.

Chapter 1

Introduction

1.1 Introduction to Intelligent Surveillance Systems

Video content analysis is more and more important for us: house care, burglarproof systems, photography and so on. Especially for surveillance systems, it usually takes us a lot of time to watch the surveillance video to get the useful information. If we want to alert an alarm immediately when emergency happens, we need to hire someone to stare at the monitors all the time. These kinds of work are laborious and time-consuming. However, the next-generation surveillance systems are different. They can real-time extract all the information that we needed. We do not have to surf all the video. All we have to do is read the information from such intelligent surveillance systems or use computers to search what we want.

Visual target tracking is an important task for many industrial applications such as intelligent video surveillance systems, human-machine interfaces, remote sensing systems and defense systems. Tracking targets in video is also a common problem in many multimedia applications, such as smart rooms, sport analysis, and video presentation enhancement. For example, we can use the tracking algorithm together with spot light machines or cameras. The spot lights and cameras can track the spokespersons automatically. Also, if we want to use the function of auto-focus in the DSC camera to focus on a target, a good tracking algorithm

is indispensable. Tracking objects together with segmentation are also important functions in the next-generation surveillance systems. However, these algorithms should be robust under several challenging conditions that happen in the real environment with surveillance systems. Moreover, the acceleration of these algorithms is also important for applications in real-time, since most of the algorithms are computationally intensive.

1.2 Tracking Algorithm and Challenges

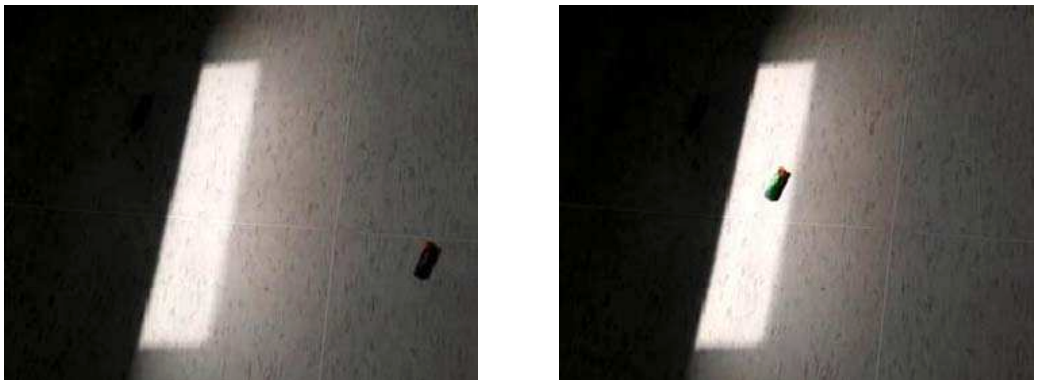
Mean-shift [1] is a fast way to track the video objects. The method models the similarity between the image regions and the target model as a cost function. Through iteratively gradient ascent on the cost function, the image region that matches the target model most is selected as the target's new position. The similarity matching is based on color histogram comparison with Bhattacharyya Coefficient. However, the mean-shift method has the local maximum or minimum problem, which means the nearby background image region that is similar to target model may be selected. In Yin et al. [2] work, they subdivide foreground and nearby background into several regions at pervious frame to build the foreground color model and background color model. On the current frame, the regions which are more similar to foreground color model and less similar to the background color model will get higher weight. After obtain this weight information as a weight image, the mean-shift process is applied on the weight image to find the region with the highest weight as the estimated object position. In [3], Yilmaz et al. fuse the color and edge information, so that their cost function contains both RGB color and shape clues. The cost function is minimized to find the object position.

The state-of-the-art algorithm is Particle Filter (or Monte Carlo method) [4] [5] [6] [7] [8] [9] estimates the object state posterior with discrete samples. Particles are spread evenly or designedly on the frame. Each of the particle is an assumption about the object state, which often includes object position and object size. Each

particle will be evaluated based on its similarity to the target model and be given a weight. The higher the similarity, the higher the weight. All of the particles or some of particles with the highest weights are used to estimate the location of the target. As for the spread of the particles , the velocity of target object is often considered [10]. With the information of velocity from previous frames, more particles are scattered at the predicted location with the velocity. Although we can get better result when we use more particles, we cannot spread too many particles by the limitation of computation resources. Pan et al. [11] address this problem by use dynamic amount of particles in each frame and variance selection to scatter the particles. As for the decision of particle weights, using color histogram to matching target object and particles is a typical way and Bhattacharyya Distance is usually used to compare the color histograms [4] [5] [6] [7] [8] [9]. Wang et al. [12] proposed a spatial-color mixture of Gaussians model instead of use color only to model the target object and the particles. They also use the probability of the Gaussian distribution of those models to be the similarity measure function to calculate the particle weights.

One challenging task in video object tracking is the occlusion problem. It occurs when there are complex interactions between targets. There are several works addressing this problem. Jiang et al. [13] [14] use a network model to model the interactions between multiple objects. Tsai et al. [15] use a 3D model to detect occlusion and use block matching to handle it. Hu et al. [16] add a new vector, which represented the occlusion relationship between objects, as a parameter into the particle weighting function.

Two other challenging tasks are related to the object appearance. One is the object appearance changes due to lighting condition changes. If an object move through a region that gets different luminance on it, such as spot light, shadow region, and so on, the appearance of the object, such as color, will change. Fig. 1.1 shows that object color change rapidly under different lighting conditions. In this condition, the tracking algorithm may converge on the wrong position. The other



(A) (B)

Figure 1.1: Object gets different colors in different luminance regions.

challenging task related to object appearance is the background-alike color on the object, which will cause the tracking algorithm converges on the similar-colored background image region and the tracker drafts. In this thesis, we proposed a particle filter framework that addresses these two problems. We propose a method to decide the particle weight rather than the typical method, i.e. color histogram comparison with Bhattacharyya Distance. The proposed method will be introduced in Chapter 2.

1.3 Media Processors and Hardware Design Challenges

The critical obstacle in multiple objects tracking is presented by the limitation of the computation resources. The similarity measurement and iterative computation take lots of time in excitation. Large memory usage is also a big problem here. Because we have to save lots of features, such as color histogram and edge histogram, and source images into the memory. Besides video object tracking, many other vision algorithm, such as video object segmentation [17] [18], feature detection and feature description [19], face detection [20], and etc, share the

same computational characteristics, which often make the implementation very challenging.

The most cost effective way to implement vision algorithms in a intelligent surveillance system is System-on-a-Chip (SoC) with all functions embedded on one chip in the camera, which is called the smart camera. Several design challenges are introduced in such an smart camera. First of all, the large computations of content analysis algorithms will increase the computing time and the area of the processing elements. Moreover, the large usage of the local memory will increase the chip area. What we have to do is designing a chip system, in which the hardware can be reused and the resource on this chip are all appropriately utilized. Moreover, the hardware should be design as a programmable hardware, since there are many algorithms and applications that should be considered to be implemented.

There are several common characteristics in the data processing of media processors: single instruction multiple data (SIMD), large amount of data transfer, high memory usage, and iterative computation. Fig. 1.2 shows the three levels of processing of media processor systems [21]. The low level is often SIMD computation of large number of pixels. It is very efficient for a parallel processing architecture. Intermediate level data processing need to be handled not only parallelly but also sequentially. Large numbers of processing elements (PE) are used to calculated large amount of data at the same time. The high level processors are just like CPUs in computers. Some MIMD can be done by this kind of processors.

Matrix [22] [23] [24] [25], which is designed by Mitsubishi, is the massive parallel processor which can process huge amount of data at same time with its 2048 fine-grained PEs and self-designed SRAM. The SRAM uses asynchronous RS-latch instead of flip-flop to reduce the maximum operation frequency and completes read-modify-write (RMW) three operations in one cycle. Xetal-II [21], designed by Philps, is also a massive parallel SIMD processor. It can handle the low level data and part of intermediate level data. It organizes its 320 PEs to 40

compute tiles, so that it can calculate some complex window-based computation. It also uses input-output stream interface to achieve serial-to-parallel and parallel-to-serial data transfer, and build a look-up-table into the chip to compute some computation such as gamma correction. Although this kind of processors can get high through-put, it cannot be used to complete difficult computation. A serious problem will be raised if the input cannot stream in as wish, and this will cause some of the PEs idle.

The PEs in Integrated Memory Array Processor for Car Electronics (IMAP-CE) [26] [27] [28], designed by NEC, have been designed to completed more tasks, such as Fourier Transform, recursive neighborhood operation, connected component labeling. Kyo et al. have analyzed and grouped those algorithms into several types and tried to port them on the PEs by handling the data transfer between PEs, which is called pixel updating line. To deal with the problem of data in and out, they design an external interface with DMA and 2KB data cache and 32KB instruction cache. The sequential data processing can be done by dealing with all of or some of the start points at the same time. After that, more and more data are prepared to be processed. Point operation, local neighborhood operation, global operation, statistical operation, geometrical operation, recursive neighborhood operation, object operation are seven main types operation IMAP-CE can complete. Stream Processor [29] [30] [31] and Reconfigurable Streaming Vector Processor (RSVP) [32] [33] [34] [35], designed by Motorola, are also this kind of processors. The chip imports the input data just like streams, and are buffered in the cache or line buffer on the chips. The central controller controls the data flow and gives the instructions to PEs. These processors do not need a lot of PEs to complete the tasks; they use the resource on the chip well, instead.

Raw microprocessor [36], designed by MIT, uses tiled architecture and on-chip networks (NoC) to achieve the needs. Every tile can be programmed differently, and the communication between tiles can be made freely with the design of NoC. Each tile can be seemed as not only a processing element but a little com-

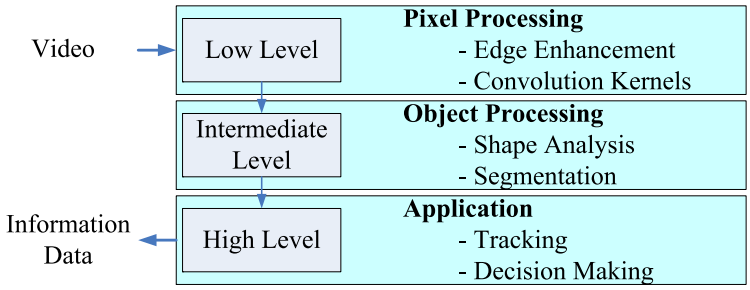


Figure 1.2: Three levels of data processing.

puter. Although this kind of processors is powerful, the cost, area, and the power are high.

Table 1.1 shows the chip specs of those media processors mentioned above. The media processors in the above show some characteristics, such as high power, large area but low hardware utilization efficiency(PE idle and then the throughput is low). If we want to get performance higher, we must design the stream of data carefully. PEs are not the more the better, but how to stream the data into the PEs and use PEs efficiently is much more important. In order to achieve lower power, lower cost, higher throughput(higher hardware utilization), and, at the same time, reserve the programmability, we design our hardware coprocessor as a reconfigurable hardware coprocessor. The relationship among ASIC, Reconfigurable Hardware, and Microprocessors is shown in Fig. 1.3. We can see that the reconfigurable hardware exhibits the desired properties. Moreover, in order to achieve more efficient data streaming, sub-word level parallelism(SLP) is also employed in the design of our hardware. The throughput can be enhanced, the bus bandwidth can be saved and the hardware sharing can be also increased with SLP. The proposed reconfigurable hardware coprocessor design is introduced in Chapter 3.

Table 1.1: The chip specs of media processors

	Raw	**Xetal-II**	**Matrix**	**RSVP**	**Stream**	**IMAP-CE**
Technology	180nm	90nm	90nm	180nm	130nm	180nm
Area	331mm2	74mm2	4.09mm2	45.53mm2	155mm2	121mm2
SRAM Size	4.8um2	10Mbit	0.99mm2	32kB	261.5kB	290kB
Frequency	425MHz	84MHz	200MHz	NA	800MHz	100MHz
Power	21W	0.6W	0.25W	NA	10.5W	2.5∼4.0W
Performance	4.8 GOPS	107 GOPS	40 GOPS	NA	512 GOPS	51.2GOPS

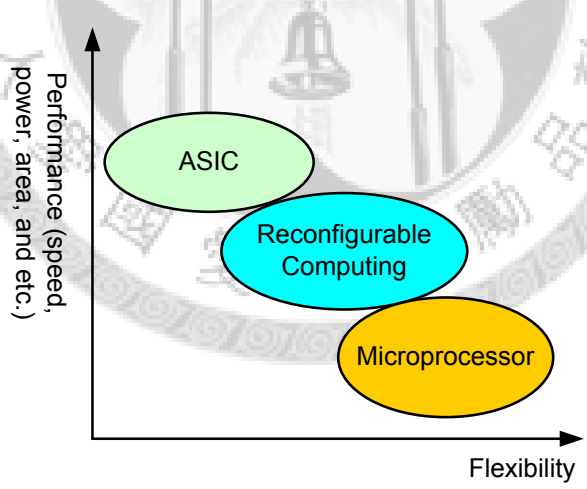


Figure 1.3: The relationship among ASICs, Reconfigurable Hardware, and Microprocessors.

1.4 Thesis Organization

In this thesis, we will focus on the tracking algorithm and designing a reconfigurable hardware coprocessor for intelligent surveillance applications that can support many vision algorithms, including the one we propose. The proposed tracking algorithm will be described in Chapter 2. In Chapter 3, the architecture design of our reconfigurable hardware coprocessor will be shown. Chapter 4 will demonstrate the experiment results of our algorithm and the implementation results of our proposed coprocessor. Finally, a conclusion is given in Chapter 5.

Chapter 2

Proposed Tracking Algorithm

2.1 Particle Filter

Particle Filter is also known as sequential Monte Carlo methods [11] [37] [38], and it is a common stochastic model and can be used to estimate the current state of some objects by the observations along time. As shown in Fig. 2.1, we try to estimate $p(X_t|Z_{(1:t)})$ by using the known $p(Z_t|X_t)$ and $p(X_t|X_{t-1})$, and the states are unknown. The figure shows a kind of Hidden Markov Model. However, the way to solve $p(X_t|Z_{(1:t)})$ must use integration which is a high computation work, so we represent the posterior probabilities by a set of random weighted samples. This way to estimate the result of integration is called Monte Carlo Integral. The more the samples, the answer is closer to true probability density function, and the samples are called ”particles” here in Particle Filter. We spread particles, calculate their weight, and use those weighted particle to estimate the posterior of the state of objects.

There are three main steps in the algorithm of Particle Filter: sampling, evaluation, and re-sampling. Sampling means that spread particles at the designed locations or random locations, and if the algorithm can spread particles well-designed, the algorithm do not need too many particles. This part of algorithm can be shown in Chapter 2.5, which is a way designed by us. We considered that the evalua-

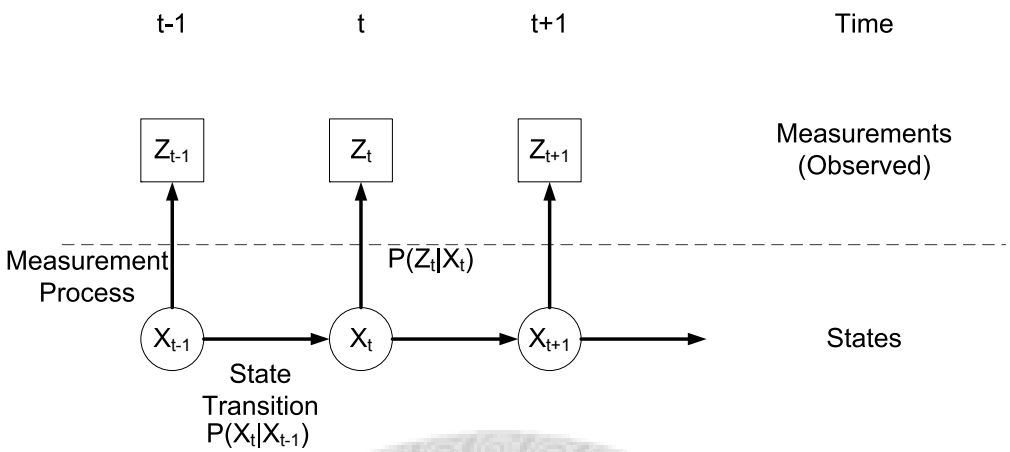


Figure 2.1: Illustration of Particle Filter.

tion function is the most important step in Particle Filter. It is used to compute the weights of particles. Efficient and precise determination of the weights are needed. We can use those weights to estimate the current state. In tracking, color histogram is most often used to be the feature to decide the particle weights or the model, so the algorithm of histogram comparison is very important. Because the tracking algorithms need to compare the difference between candidate image regions and target objects' models, the histogram comparison function dominate whether the tracking algorithms are good or not. We can say that the design of evaluation function is the kernel of tracking algorithms which use Particle Filter. The proposed method of histogram comparison are described in Chapter 2.7. The typical histogram comparison methods are compared with the proposed one. The computation of the particle weight is described in Chapter 2.8, and we also combine the result of segmentation. The third step is re-sampling, and it uses the weights calculated in evaluation step to re-sample the particles, and prepares the particles for next iteration.

For example, when we use Particle Filter in the tracking algorithm, the weight of a particle is the likelihood that the particle and the target object are the same. The higher weight means that the particle is more similar to the target object. The

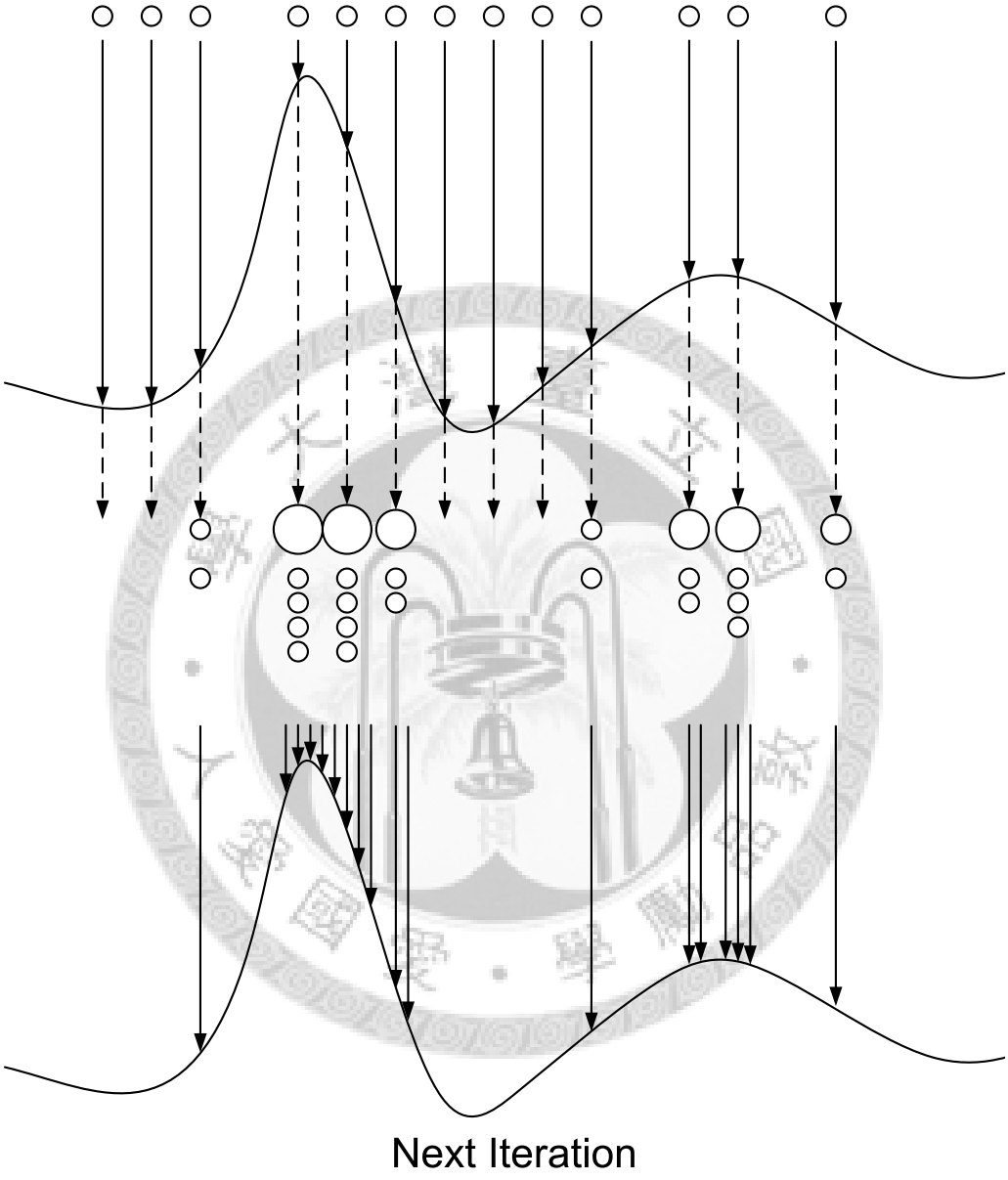


Figure 2.2: A sample of re-sampling.

position of the particle that has the global maximum of the weight is the estimated position of the target object.

Let state X_t represents the target's state, such as position or size, and Z_t is the observations at time t. In ideal Monte Carlo sampling, N independent and identically distributed samples, or particles, can be sampled. Particles S_t^k and corresponding weight w_{t^k} can be used to estimate the posterior density function $p(X_t|Z_{1:t})$.

$$p(X_t|Z_{1:t}) \approx \sum_{i=1}^{N} w_t^k \delta(X_t - S_t^k) \tag{2.1}$$

The formula of calculating weight of particles is shown below

$$w_t^k \propto p(Z_t|X_t = S_t^k) \tag{2.2}$$

or in [38]

$$w_t^k \propto \frac{p(S_{0:t}^k|Z_{1:t})}{q(S_{0:t}^k|Z_{1:t})} \tag{2.3}$$

$$w_t^k = w_{t-1}^k \frac{p(Z_{1:t}|S_{0:t}^k)p(S_{0:t}^k|S_{0:t}^k)}{p(S_{0:t}^k|S_{0:t-1}^k, Z_{1:t})} \tag{2.4}$$

$q(.)$ is the importance function, or a density function, and is used to generate samples. Eq. 2.4 is the weight update formula. To simplify the formula, the Eq. 2.2 is often used to calculate the weight, and this part of calculation will be mention in Chapter 2.8 as step 6 in Fig. 2.3. The normalized weight π_t^k is given by

$$\pi_t^k = \frac{(w_t^k)}{\sum_{j=1}^{N} w_t^j} \tag{2.5}$$

After weight normalized, the estimated state O_t is given as the formula bellow

$$O_t = \sum_{j=1}^{m} (w_t^j)'(S_t^j)' \tag{2.6}$$

$(w_t^j)'$ and $(S_t^j)'$ are the sorted weight and particles. Top m samples is used to estimate the state. After the estimated state is gotten, the particles is re-sampled.

2.2 Tracking Algorithm Overview

2.2.1 Design Challenge

There are two common problems in tracking algorithm today: trackers may draft under object appearance changes due to lighting condition change, and trackers may draft in similar-colored background. The first problem is caused by the histogram of target object will change after the change of object appearance due to lighting condition change. The traditional method of comparison of histogram in tracking algorithm cannot handle this kind of problem. It is important to employ a new histogram comparison method to overcome this problem.

The second problem is caused by the tracker cannot discriminate the histogram between target object and background. The clue of appearance may be less important. How the algorithm can extract the useful clues, and how it can discriminate between object and background is the problem the algorithm has to solve.

2.2.2 Tracking Algorithm Overview

The proposed algorithm does not need a training step to setup the target model or users to startup the tracking function. A segmentation algorithm is used to know whether moving objects come in or not and where they are. Connected component labeling is also used to cut several foreground regions out and remove some foreground regions whose area are too small. After that, the remaining foreground regions are the target objects. These target objects are not restricted to be a specific object, and they can be cars, humans, animals, and so on. In the algorithm all large enough moving objects can be tracked. Even more, if someone drives a car in, and then get off the car, a new tracker will be produced to tracker the guy.

A simple flow chart is shown in Fig. 2.3. At step 1, the segmentation algorithm is performed to segment the foreground out. and connected component labeling is used to draw the bounding boxes of foreground regions. Those bounding boxes'

size and position are stored into the object stack. At step 3, the proposed spreads particles by Gaussian distribution, and accumulates the histogram of every particle at step 4. Then the algorithm compares the histogram between target object and particles. At step 6, the proposed uses the white points made at step 1 and the histogram comparison result made at step 5 to get the weights of all particles, and use them to estimate the position of target objects at step 7. The proposed also update the information of target objects. The proposed adds new trackers if there are some objects in object stack and they are large enough but no tracker is tracing it. The proposed initializes the new tracker and add it into next frame tracker stack at step 11.

The proposed does not use 3-D histogram but three 1-D histograms instead, because the computation time and the memory usage are too large. 3-D histogram need to store $256^3 = 16777216$ bins data, but three 1-D histograms need to store $256 \times 3 = 768$ bins data. Also, the computation time of histogram comparison between two 3-D histograms is much larger than that between two 1-D histograms, and this part will be discussed in Chapter 2.7.2.

2.3 Segmentation

Chan's work [17] is implemented in segmentation. When a pixel from current frame comes into the segmentation core, it is compared with the background model. If they are matched, that is, the difference between the current frame pixel and the background pixel is lower than a threshold BD_{th}, this current frame pixel is background, and the matched background weight increases. 4 layers of background are used, and 4 background layers are compared to decide whether the current frame pixel is background or not. If the difference is larger than the threshold, the background weight will decrease. If the difference between the pixel and all the background are all larger than the threshold, the current pixel is classified as foreground.

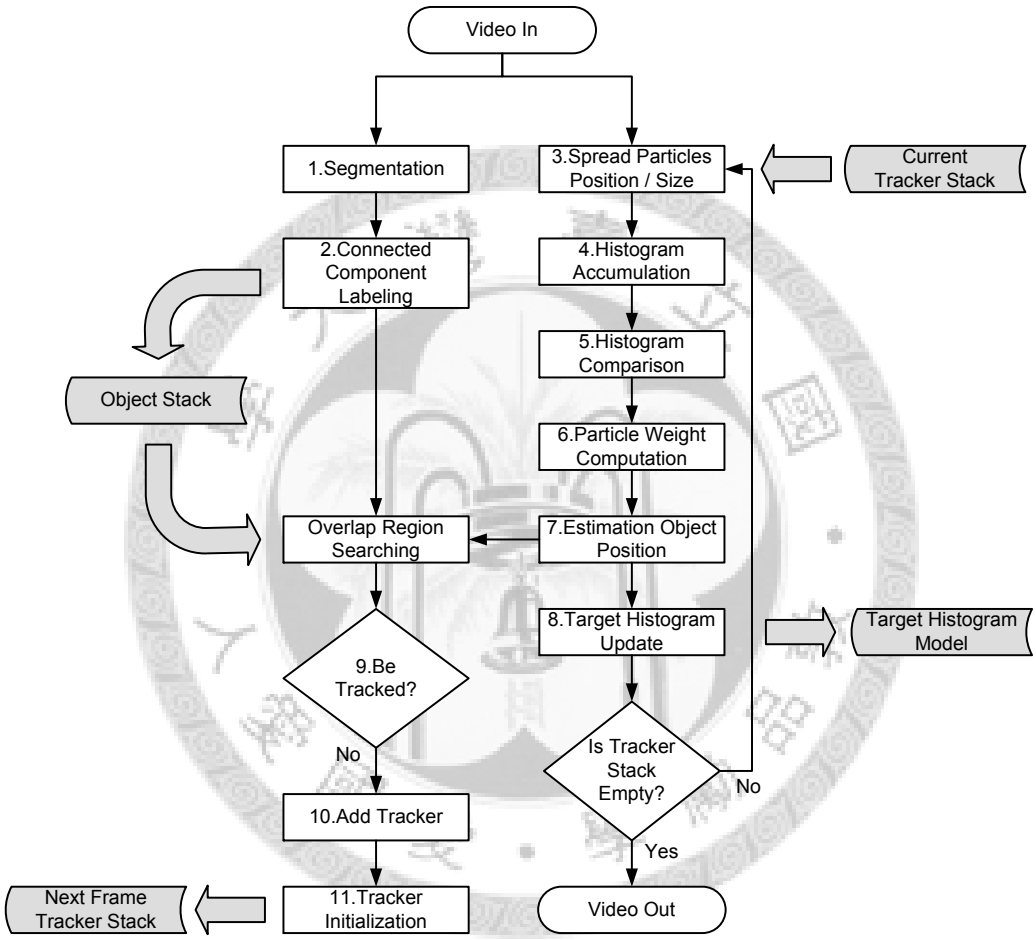


Figure 2.3: Tracking algorithm overview

2.4 Connected Component Labeling

Because it has to be known that which foreground regions are connected to be one object, connected component labeling is used to identify which regions are connected. The area of all connected regions can also be calculated, and the regions whose area are too small is removed, because those small moving object may be some noise or some meaningless objects, such as moving leaves on the tree.

4-connectedness is used in the connected components labeling algorithm. Several iterations are needed. At the first iteration, we have to give an ID to every pixel which belongs to foreground and give 0 to every pixel which belongs to background. The algorithm starts from the top-left pixel and then right and then down (raster scan). The pixel ID is the max value of left pixel ID and bottom pixel ID. If both ID are 0, the algorithm gives the ID as the same value as an ID counter, and adds 1 to the ID counter. After this iteration, every pixel will be assigned an ID.

At the second iteration, the similar thing is performed but do it bottom-up. The algorithm starts from the bottom-right pixel and then left and then up. The pixel ID is modified to be the max value of right pixel ID and up pixel ID. At the third iteration, the algorithm does the top-down iteration again. After several iterations, all regions that are connected together are assigned the same IDs.

After the regions whose area are too small (the area is smaller than 0.5% of frame size) are removed, a mapping table is used to make the regions' IDs continuous. Because after connected component labeling, it cannot be known that which ID is assigned to the region. The mapping table is used to map the IDs. After that, bounding boxes are drawn on those regions, and the coordinates of the vertex are stored at the memory. Those coordinates will be the initial parameters of the target objects.

2.5 Spread Particles

It is important to spread particles efficiently because the algorithm cannot spread too many particles and it costs too much.

Estimating the location where the target is more likely at is needed, and the algorithm spreads particles near the location according to a Gaussian distribution. There are four variables in the particle: x-coordinate ($S^k_{x,t}$), y-coordinate ($S^k_{y,t}$), height ($S^k_{h,t}$), and width ($S^k_{w,t}$). If the proposed wants to decide the four variables at once, it has to spread points in a 4-dimensional space. It is a high-computational work to do that, so the proposed separates the computation into two 2-dimensional spaces. That is, the proposed spreads points in position space and size space separately.

The proposed spreads particles in two passes: 50 particles to decide the position of the object, and 50 particles to decide the size of object. That is, the flow of the particle filter tracking in Fig. 2.3 is executed twice for a target object in a frame. At the first pass, 50 particles are spread in different positions but the same size. First, a velocity function is developed, as following formula

$$v_{x-1} = \frac{O_{x,t-1} - O_{x,t-3}}{2} \tag{2.7}$$

and

$$v_{y-1} = \frac{O_{y,t-1} - O_{y,t-3}}{2} \tag{2.8}$$

to estimate the velocity of the target object. The position of particles can be calculated by the formula below

$$S^k_{x,t} = O_{x,t-1} + v_{x-1} + N(0,5) \tag{2.9}$$

and

$$S^k_{y,t} = O_{y,t-1} + v_{y-1} + N(0,5) \tag{2.10}$$

$S^k_{x,t}$ is the x-coordinate of sample k, and $S^k_{y,t}$ is the y-coordinate of sample k. $N(\mu, \sigma^2)$ is the normal distribution with the mean μ and the variance σ^2, and

$0 \leq k \leq 49$. At this pass, the sizes of particles are as large as the object size estimated at previous frame.

At the second pass, we spread 50 particles in different size but the same position. Following is the formula of spreading particles in different size,

$$S_{h,t}^{k} = O_{h,t-1} \times [1 + (U(-4,4))/35] \tag{2.11}$$

and

$$S_{w,t}^{k} = O_{w,t-1} \times [1 + (U(-4,4))/35] \tag{2.12}$$

$S_{w,t}^{k}$ is the width of sample k, and $S_{h,t}^{k}$ is the height of sample k. $U(a,b)$ is the uniform distribution between a and b, and $0 \leq k \leq 49$. At this pass, the positions of particle are the same as the positions estimated at the first pass.

2.6 Histogram Accumulation

As mentioned above, the proposed does not accumulate one 3-D histogram but three 1-D histogram instead. The appearance histogram of all particles is accumulated by the formula bellow

$$\begin{cases} q_y(Y) = \sum_{x=c_x-w/2}^{c_x+w/2} \sum_{y=c_y-h/2}^{c_y+h/2} \Delta(x,y) \cdot \delta[I_y(x,y) - Y] \\ q_u(U) = \sum_{x=c_x-w/2}^{c_x+w/2} \sum_{y=c_y-h/2}^{c_y+h/2} \Delta(x,y) \cdot \delta[I_u(x,y) - U] \\ q_v(V) = \sum_{x=c_x-w/2}^{c_x+w/2} \sum_{y=c_y-h/2}^{c_y+h/2} \Delta(x,y) \cdot \delta[I_v(x,y) - V] \end{cases} \tag{2.13}$$

(c_x, c_y) is the centroid of the particle. $I_y(x,y), I_u(x,y), I_v(x,y)$ are the Y, U, and V pixel value of the point (x,y), w is the width of the particle, and h is the height. The histograms of all three channels are accumulated separately, for the purpose of less memory usage. There are two modes can be used to accumulate histogram: centroid emphasizing mode and uniform accumulation mode. Because the proposed would like to pay much attention to the color on the object and pay less attention to the color on the background, the proposed should give much weight to the color in the center of the particle. The object is more likely at the

middle of the particle. The formula bellow is used to describe this

$$\Delta(x,y) = \frac{diag - \sqrt{(x-c_x)^2 + (y-c_y)^2} + 1}{Sumofhistogram} \tag{2.14}$$

1 is added in Eq. 2.14 is to avoid $\Delta(x,y)$ become 0. As, Fig. 2.4 shown, diag is the half length of the diagonal line of the particle. Sum of Histogram is the sum of the histogram of the particle, and it is used to normalize histogram.

When (x,y) is closer to the center of particle, the higher weight it gets. The second mode is the uniform accumulation mode, and the formula of $\Delta(x,y)$ is shown bellow

$$\Delta(x,y) = \frac{1}{Sumofhistogram} \tag{2.15}$$

The Sum of Histogram here is equal to the area of the particle.

For emphasizing the foreground color more, Equation (3.12) is modified by the result of segmentation as following formula.

$$\begin{cases} q_y(Y) = \sum_{x=c_x-w/2}^{c_x+w/2} \sum_{y=c_y-h/2}^{c_y+h/2} \Delta(x,y) \cdot \delta[I_y(x,y) - Y] \cdot choose_y(x,y) \\ q_u(U) = \sum_{x=c_x-w/2}^{c_x+w/2} \sum_{y=c_y-h/2}^{c_y+h/2} \Delta(x,y) \cdot \delta[I_u(x,y) - U] \cdot choose_u(x,y) \\ q_v(V) = \sum_{x=c_x-w/2}^{c_x+w/2} \sum_{y=c_y-h/2}^{c_y+h/2} \Delta(x,y) \cdot \delta[I_v(x,y) - V] \cdot choose_v(x,y) \end{cases} \tag{2.16}$$

and

$$\begin{aligned} choose_y(x,y) &= \begin{cases} 0, & if\, bg_y(I_y(x,y)) > Th_{bg} \\ 1, & else \end{cases} \\ choose_u(x,y) &= \begin{cases} 0, & if\, bg_u(I_u(x,y)) > Th_{bg} \\ 1, & else \end{cases} \\ choose_v(x,y) &= \begin{cases} 0, & if\, bg_v(I_v(x,y)) > Th_{bg} \\ 1, & else \end{cases} \end{aligned} \tag{2.17}$$

bg_y, bg_u, bg_v are the histogram of the background color, and Th_{bg} is the threshold value. After the proposed segments the foreground and background at the Step 1 in Fig. 2.3, background images BG can be gotten. The background histograms

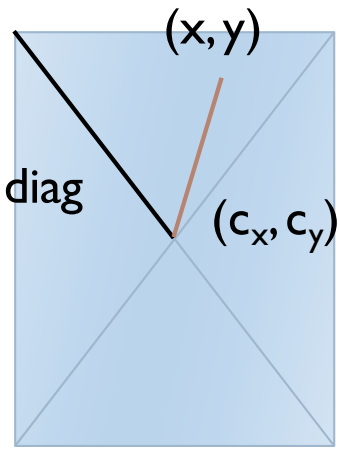


Figure 2.4: Center emphasizing mode

bg_y, bg_u, bg_v at the position of the particles are accumulated.So that the histogram of the background color are calculated for Eq. 2.17. If the bin value of one color in background histogram is larger than the threshold Th_{bg}, the color will be seen as a background main color and will not be accumulated in the foreground histogram.

2.7 Histogram Comparison Algorithm

The main thing we want to do here is to know which histogram is much similar to the target histogram. This step is the most important part in Particle Filter algorithm as mentioned above. Because every particle of every target object of every frame should be calculated once, a fast algorithm has to be found out, and an accurate and efficient algorithm should be developed to calculate this. The proposed algorithm should be able to handle the problem of the light condition change, so the algorithm is needed to be a robust one. To be robust, those histograms have to be normalized first to handle the change of the particles' size.

2.7.1 Cross-bin And Bin-by-bin Computation

There are two kinds of histogram comparison algorithms : cross-bin and bin-by-bin. Assume there are two histograms $P = \{p_i\}$, $Q = \{q_i\}$, where p_i and q_i are their bin values and they are all normalized, respectively. The bin-by-bin methods only compare contents of corresponding histogram bins, that is, they only compare p_i and q_i for all i, but not p_i and q_j for $i \neq j$. The cross-bin methods also contain the terms that compare non-corresponding histogram bins.

The cross-bin methods consider the distance between two bins, but the bin-by-bin methods do not. As prediction, bin-by-bin methods may break down when color vary, but cross-bin methods may be robust to scale and appearance change.

Bhattacharyya Distance

This is one of the bin-by-bin methods and proposed by A. Bhattacharyya [39] [40]. It is widely use in histogram comparison because it is easy to implement and the its complexity of timing is about $O(M)$. M is the number of histogram bins. It is defined as

$$D(P,Q) = -ln(BC(P,Q)) \tag{2.18}$$

where

$$BC(P,Q) = \sum_{i=1}^{M} \sqrt{p_i q_i} \tag{2.19}$$

It only compares corresponding histogram bins. The distance can be calculated when the histogram are being accumulated with the following formula

$$D(P,Q,t) = \sum \sqrt{p_i q_i} = \sum \sqrt{\frac{p'_i}{(Total\ Histogram)} \cdot q_i} \tag{2.20}$$

p'_i is the un-normalized histogram, and p_i and q_i are the normalized histogram. $D(P,Q,t)$ is the temporal distance at time t. Now a new D' can be defined

$$D'(t) = \sqrt{(Total\ Histogram)} \cdot D(t) = \sqrt{p'_i \cdot q_i} \tag{2.21}$$

When the algorithm wants to add new data β in bin b, at time t+1

$$D'(t+1) = D'(t) - \sqrt{p'_m q_m} + \sqrt{(p'_m + \beta) \cdot q_m} = D'(t) + \sqrt{q_m}(\sqrt{(p'_m + \beta)} - \sqrt{p'_m}) \tag{2.22}$$

and

$$D(t+1) = \frac{D'(t+1)}{\sqrt{Total\ Histogram + \beta}} \tag{2.23}$$

The computation time can be hidden into the time of accumulating histogram. Bhattacharyya is a measurement of the amount of overlap of two statistical distributions. However, the Bhattacharyya distance will be 0 if there is no overlap at all due to the multiplication by zero in every bin, and this means the distance between fully separated histograms will not be exposed by this coefficient alone.

Earth Mover's Distance

This is one of the cross-bin methods and proposed by Rubner et al. [41] [42]. It is a metric to evaluate dissimilarity between two multi-dimensional distributions in some feature space and it is applied as a solution to the well-known transportation problem. That is, there are many stacks of mounds with different amounts, and also many holes with different capacities. The transportation problem is that if we want to fill all the holes with the mounds, what is the effort we should spend?

This problem can be changed to the problem of histogram comparison. If the algorithm wants to compare histogram P, m bins, and histogram Q, n bins, it sets p_i as mounds and q_i as holes, and flow $F = \{f_{ij}\}$ with f_{ij} representing the flow between p_i and q_j. $D = \{d_{ij}\}$ is the ground distance between p_i and q_j. What we want to do is minimize

$$EMD(P,Q,F) = \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} f_{ij} \tag{2.24}$$

Subject to the following constraints,

$$f_{ij} \geq 0, 1 \leq i \leq m, 1 \leq j \leq n \tag{2.25}$$

$$\sum_{j=1}^{n} f_{ij} \leq Amount\ of\ p_i = w_{p_i} \tag{2.26}$$

$$\sum_{i=1}^{m} f_{ij} \leq Capacity\ of\ q_j = w_{q_j} \tag{2.27}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} = min(\sum_{i=1}^{m} w_{p_i}, \sum_{j=1}^{n} w_{q_j}) \tag{2.28}$$

Eq. 2.25 restrict that it is not possible to move earth from holes to mounds, and Eq. 2.26 and Eq. 2.27 shows that a mound cannot become a hole or a hole cannot become a mound. The histograms are both be normalized, so that

$$\sum_{i=1}^{m} w_{p_i} = \sum_{j=1}^{n} w_{q_j} \tag{2.29}$$

It is very effective for distribution with sparse structures. However, the time complexity is about $O(n^3)$ where n is the number of histogram bins. The problem can be described as a matrix problem, and can be solved by linear algebra.

Diffusion Distance

It is proposed by Ling et al. [43], and simulate the histogram comparison process as the temperature diffusion in an isolated field. The heat diffusion equation is used to estimate the temperature near some heat source at some time. A new histogram by subtracting the bin value of target histogram from the bin value of candidate histogram at the same bin should be produced at the first place. It can be imagined that the bin value of the new histogram is the temperature at the position of the bin. The heat will conduct from high temperature to low temperature, and the heat diffusion equation can get every temperature at every position. The sum of the absolute value of these temperatures are considered as the result of histogram comparison, or distance. If the high temperature region is close to the low temperature region, the distance is small; otherwise, the distance is large. That is, small distance can be gotten if the target histogram is similar to the candidate histogram, which may be derived from the target histogram with some small positional shifting of the histogram bin.

They simplify the heat diffusion equation and derive the diffusion distance as

$$D(P,Q) = \sum_{l=0}^{L} (\sum_{i=0}^{M} |d_l(i)|) \tag{2.30}$$

L is the number of pyramid layers and M is the bin number of histograms, that is, M = m = n, and

$$d_0(i) = p_i - q_i \tag{2.31}$$

$$d_l(i) = [d_{l-1}(i) * \phi(i,t)] \downarrow_2 \quad l = 1, \cdots, L \tag{2.32}$$

$$\phi(i,t) = \frac{1}{(2\pi)^{\frac{1}{2}}t} e^{\frac{-i^2}{2t^2}} \tag{2.33}$$

ϕ is the Gaussian filter and t is a variable that choose the "time" of heat diffusion, or the standard deviation for the Gaussian filter. "$\downarrow_2$" denotes half size down-sampling.

The time complexity is about $O(M \times L \times WindowSize)$, M is the number of histogram bins, L is the number of layers, and WindowSize is the window size of ϕ. The WindowSize = 7 (-3 to 3) and L = 5 are chosen in our implementation. This algorithm is just like a window operation and appropriate to be implemented in hardware.

2.7.2 Comparison

Some simple example are calculated to compare the three algorithms above. Some simple histograms show in Fig. 2.5, and the numbers below the lines are the bin indexes and the numbers above the histograms are the histogram value. We may think that the difference between (A.1) and (A.2) is a little larger than (B.1) and (B.2), and (C.1) and (C.2) is smaller than (D.1) and (D.2), and (A.1) and (A.2) is smaller than (D.1) and (D.2), as shown in Table 2.2. The three algorithms are used to compare those histograms and get the data as Table 2.1. In Bhattacharyya Distance, A is larger than B because if the two histograms we compare are not overlap, the Bhattacharyya Distance is infinite or a large number here. C and D cannot be compared because they are all infinite by using Bhattacharyya Distance. In Earth Mover's Distance, C is equal to D because the distance between 10 and 15, 15 and 20, 20 and 25 are the same. This causes a serious problem when we use

Earth Mover's Distance. The comparison of these three algorithms can be shown in Table 2.2.

Table 2.1: The result of histogram comparison

	Bhattacharyya Distance	Earth Mover's Distance	Diffusion Distance
A	inf.	2	0.12328
B	0.69315	1	0.06
C	inf.	5	0.14375
D	inf.	5	0.1475

Table 2.2: The result of histogram comparison with answer

Answer	Bhattacharyya Distance	Earth Mover's Distance	Diffusion Distance
A > B	A > B	A > B	A > B
C < D	NA	C = D	C < D
A < D	NA	A < D	A < D

As shown in Table 2.3, the computation complexity of EMD is about $O(n^2)$ when the fast algorithm EMD-L1, which is proposed by Ling et al [44] are used. The *WindowSize* and the number of layers in diffusion distance are fixed in our implementation. The computation complexity of EMD is too large to be implemented in hardware. Considering the memory usage and the computation complexity, we use 1D histogram comparison only. The following formula is used to compute the total distance by making use of the distance from Y, U, and V (D_Y, D_U, D_V).

$$D = \sqrt{D_Y^2 + D_U^2 + D_V^2} \tag{2.34}$$

The three channels can be separated because if the distance between two 3D histograms is large, at least one distance from Y, U, or V must be large too. The

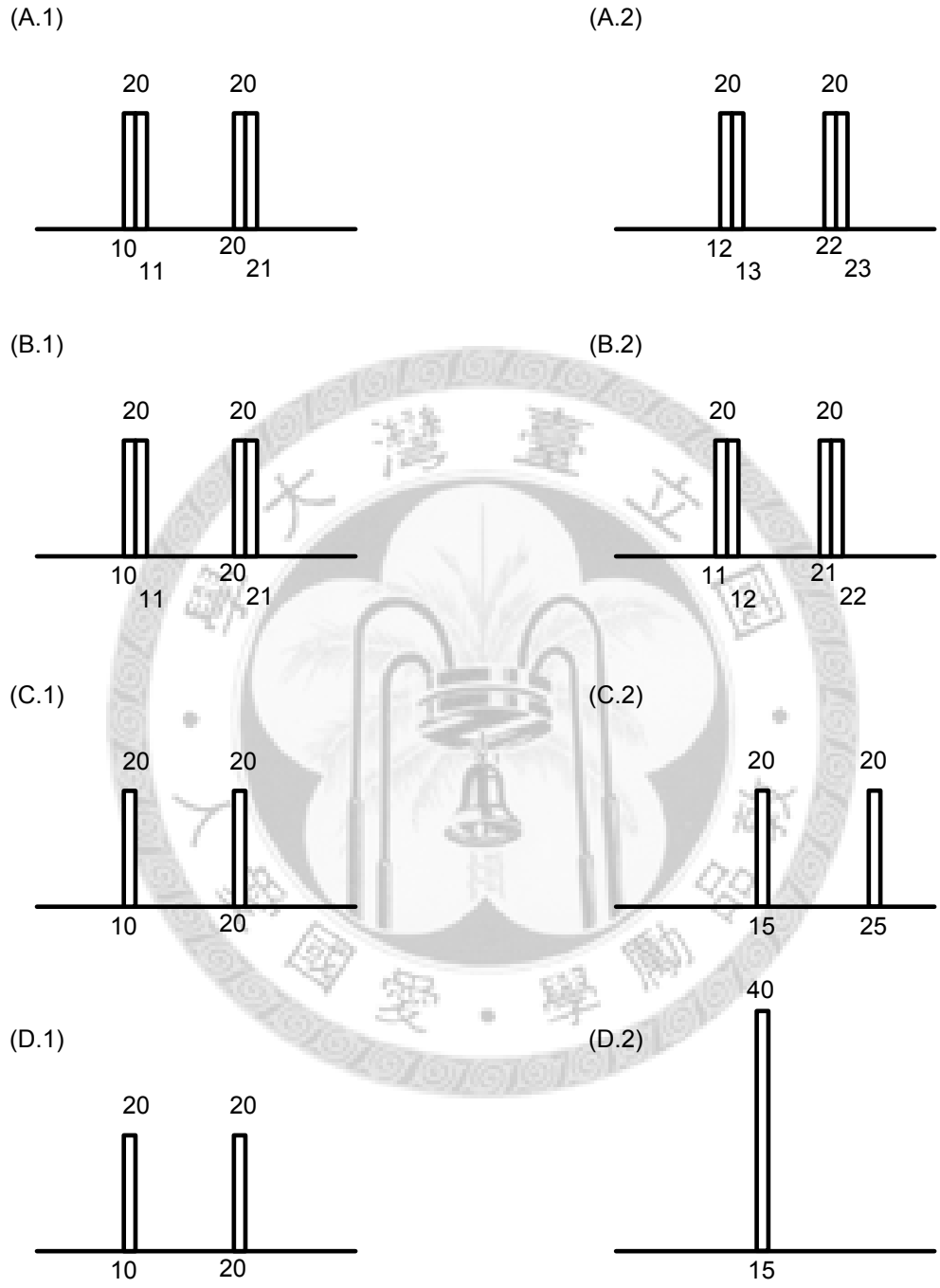


Figure 2.5: An example of histogram comparison

histogram in 3D can be imagined as points in a YUV color space, and the histogram in 1D can be imagined as the projection of the 3D histograms to the three axes. The large distance between two histogram means that the points in YUV color space are far away, so at least one of the 1-D distance must be large. It is only needed to know which distance of histograms is smaller, and the absolute value of the distance is less important.

We use a test sequence to test these three algorithms, as shown in Fig. 2.6. A guy walks through a shadow and a yellow light condition, so the color of the guy change to black, then change to yellow, and change back at final. The color box in the frame is our tracking box, and different color of boxes mean that they are considered as different objects by the algorithm. The first column is 3D Bhattacharyya Distance, that is, in this column we use $256 \times 256 \times 256$ bins to accumulate and compare the histograms. The second column is 1D Bhattacharyya Distance, the third column is Diffusion Distance, and the fourth column is Earth Mover's Distance. At start in (A), the results of four algorithms are the same. However, the tracking box in Bhattacharyya Distance is not stable as we can find out in (B). When the guy start to walk through a shadow, the tracker in 1D Bhattacharyya Distance lost in (D), but the tracker in Diffusion Distance is still good. Then algorithm adds a new tracker automatically in Bhattacharyya Distance in (E), but no new tracker in Diffusion Distance is need to added. The 3D Bhattacharyya Distance may get wrong in (F), because the guy is turning yellow but the wall is not as yellow as the guy is. When the guy walk through the yellow light, the tracker still stay in the yellow light region and causes the result bad in Bhattacharyya Distance in (G). The result of Earth Mover's Distance is as good as the result of Diffusion Distance. However the cost of Earth Mover's Distance and 3D Bhattacharyya Distance are too big. To consider the performance and the cost, we use Diffusion Distance to be the histogram comparison algorithm.

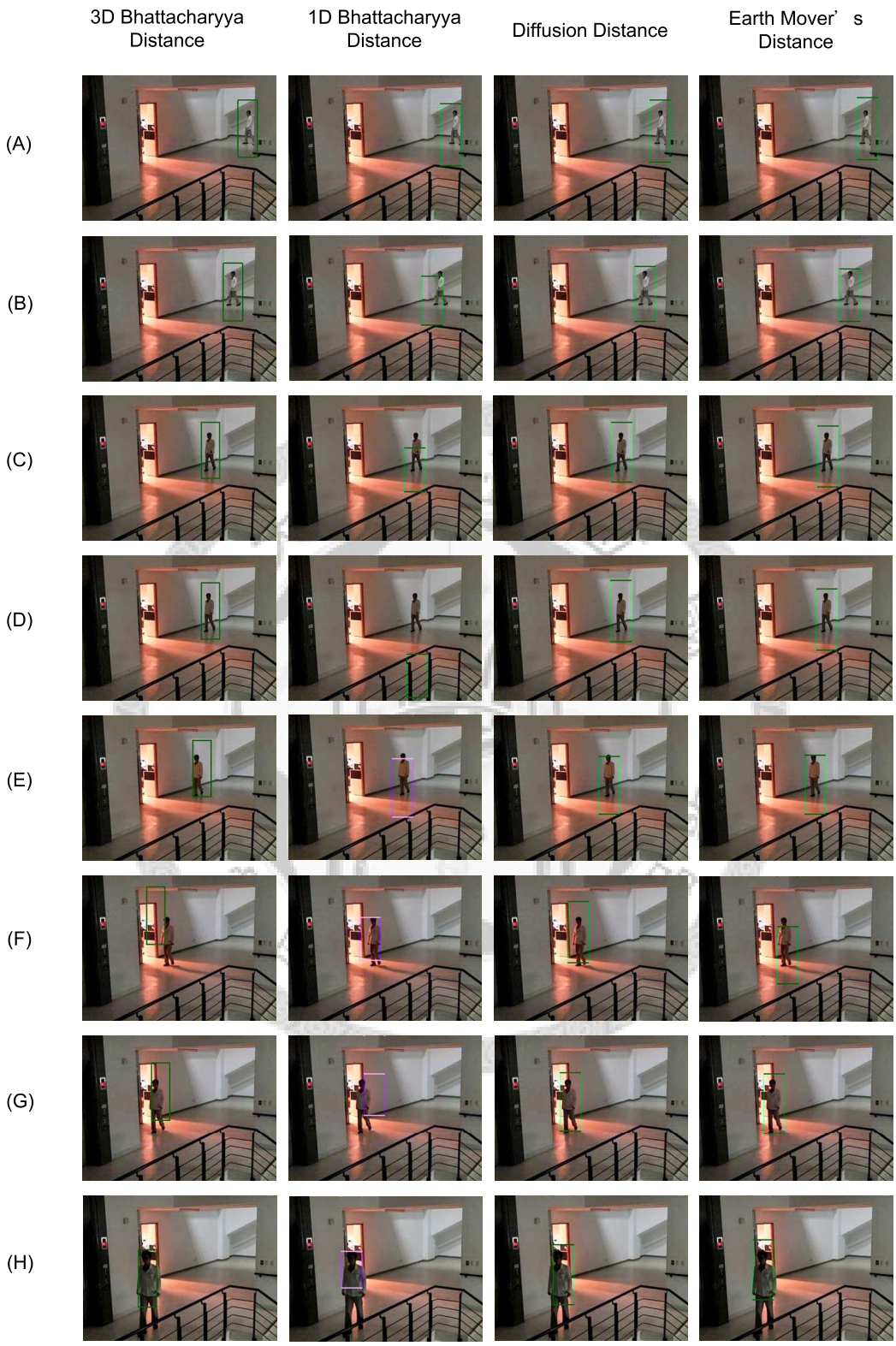


Figure 2.6: Comparison of three algorithms. The first column is 3D Bhattacharyya Distance, the second column is 1D Bhattacharyya Distance, the third column is Diffusion Distance, and the fourth column is Earth Mover's Distance. (A) frame no.8 (B) frame no.40 (C) frame no.110 (D) frame no.112 (E) frame no.149 (F) frame no.218 (G) frame no.249 (H) frame no. 340

Table 2.3: The computation complexity of three algorithms. n is the bin number of histogram in one color

	Bhattacharyya Distance	Earth Movers Distance	Diffusion Distance
1D	$O(n)$	$O(n^2)$	$O(n)$
3D	$O(n^3)$	$O(n^6)$	$O(n^3)$

2.8 Particle Weight Computation

Mapping the result of "histogram comparison" to "weight of particle k" is needed. The followed formula is developed

$$HDist_t^k = e^{-\frac{D_t^k}{GaussianKernel_t}} \tag{2.35}$$

where D_{tk} is the result of histogram comparison from Chapter 2.7. $GaussianKernel_t$ is used to restrict HDist not too large. If some of the weights of the particles are too large, the local minimums will dominant the result of tracking. $GaussianKernel_t$ can be gotten by the formula

$$GaussianKernel_t = \frac{\sum_{k=1}^{N} HDist_{t-1}^k}{GK \cdot N} \tag{2.36}$$

GK is a constant that sets the importance of the result of histogram comparison, we set it as 5 in our experiment.

The HDist is calculated like this because the proposed has to emphasize on the particle whose histogram is closer to the histogram of target object. However, the difference of D_t^a and D_t^b for $a \neq b$ is small.The difference of the weights of two different particles has to be amplified, so D_t^k is set as the exponent in Eq. 2.35. Because the value of D_t^k is not too small, it will let the result indistinct. It is divided in the Eq. 2.35.

After calculate the weight from histogram comparison, the algorithm combine the information from segmentation. The tracking algorithm can be more robust. The way the proposed combines them together can be shown as following formula

$$w_t^k = HDist_t^k + \alpha \times (Count_t^k)/(Area_t^k) \tag{2.37}$$

where α is the importance of the result of segmentation, and $Count_t^k$ is the number of the pixels which are marked as "foreground" in particle k. $Area_t^k$ is the area of particle k, that is $S_{h,t}^k \times S_{w,t}^k$. In Eq. 2.37, we use $(Count_t^k)/(Area_t^k)$ to calculate the confidence of foreground inside particle k. If the appearance of a particle is similar to the target object, and the confidence that foreground is inside this particle is high, the weight of particle is high. However, we cannot use the result of segmentation only. If the background color varies with time, result of the segmentation is not good. So, they are combined together.

2.9 Object State Estimation

Eq. 2.6 is used to estimate the position of object. The proposed cannot just use the particle which has the largest weight, because it may be the local maximum. The proposed chooses the top several particles to handle this. However, if all the particles are chosen, the algorithm may not handle the rapidly change of object. The one who has a small weight but the position is far from the correct position will still affect the result of estimated object position. In our implementation, the proposed only choose the top 40 particles to estimate the object position.

2.10 Target Histogram Update

This step is important because it is not needed to get a training step to setup the target model. The target models may not be good if the proposed just use the initial model when objects come into the frame. The target model should be updated all the time to make sure the model is good and robust. The method the proposed uses is easy and fast.

The object position $(O_{x,t}, O_{y,t})$ and size $(O_{w,t}, O_{h,t})$ can be used to accumulate a new histogram L_p_y, L_p_u, L_p_v and update the target.

$$\begin{cases} L\_p_y(Y) = \sum_{x=O_{x,t}-\frac{O_{w,t}}{2}}^{O_{x,t}+\frac{O_{w,t}}{2}} \sum_{y=O_{y,t}-\frac{O_{h,t}}{2}}^{O_{y,t}+\frac{O_{h,t}}{2}} \Delta(x,y)\cdot\delta[I_y(x,y)-Y]\cdot Mask(x,y) \\ L\_p_u(U) = \sum_{x=O_{x,t}-\frac{O_{w,t}}{2}}^{O_{x,t}+\frac{O_{w,t}}{2}} \sum_{y=O_{y,t}-\frac{O_{h,t}}{2}}^{O_{y,t}+\frac{O_{h,t}}{2}} \Delta(x,y)\cdot\delta[I_u(x,y)-U]\cdot Mask(x,y) \\ L\_p_v(V) = \sum_{x=O_{x,t}-\frac{O_{w,t}}{2}}^{O_{x,t}+\frac{O_{w,t}}{2}} \sum_{y=O_{y,t}-\frac{O_{h,t}}{2}}^{O_{y,t}+\frac{O_{h,t}}{2}} \Delta(x,y)\cdot\delta[I_v(x,y)-V]\cdot Mask(x,y) \end{cases} \tag{2.38}$$

$Mask(x,y)$ is the segmentation result, and $Mask(x,y) = 1$ when pixel (x,y) is foreground. That is, only the pixel that is foreground is accumulated, so that, the proposed does not accumulate background color into the target histogram. By using the formula above, histograms $L_p\_y(Y), L_p\_u(U), L\_p_v(V)$ can be gotten, and the original target model is updated by a learning rate γ as the following formula

$$\begin{cases} p_{y,t}(Y) = (1-\gamma)\cdot p_{y,t-1}(Y) + \gamma\cdot L\_p_y(Y) \\ p_{u,t}(U) = (1-\gamma)\cdot p_{u,t-1}(U) + \gamma\cdot L\_p_u(U) \\ p_{v,t}(V) = (1-\gamma)\cdot p_{v,t-1}(V) + \gamma\cdot L\_p_v(V) \end{cases} \tag{2.39}$$

where $p_{y,t-1}$, $p_{u,t-1}$, $p_{v,t-1}$ are the target histograms at the previous frame, and $p_{y,t}$, $p_{u,t}$, $p_{v,t}$ are the target histograms at the current frame. If a larger γ is chosen, the change of target object can be handled easily, but error may happen when the estimated object is not good enough. On the contrary, if a smaller γ is chosen, the result may get worse because of the appearance change of target object.

2.11 New Tracker Initialization

After the connected component labeling has been done, every position of all foreground objects can be gotten. If some of those foreground objects overlap large enough area of some trackers' object box $(O_{x,t}, O_{y,t}, O_{w,t}, O_{h,t})$, those objects can be said that they have been "tracked," and new trackers will be added to track those objects which have not been tracked. As a result, it is found out that the result of segmentation is not so good sometimes. Some objects may be broken to pieces. If the thing mentioned above has been done, those broken objects will be

Figure 2.7: Bonding box of a object

a problem. For example, a tracker may be added to the body of a person, and a tracker to his or her shoes, but that is not what we want. The proposed does not just use the object boxes to discriminate whether this object is tracked or not, but uses two-time larger boxes, which are called no new tracker regions, instead of the object boxes.

The proposed does not use the bounding boxes of connected component labeling to be the parameters of new trackers directly, because the bounding boxes may be too large and may contain too much background in it. As shown in Fig. 2.7, if the object is not upright or the person opens his or her arms and legs widely, the bounding box may contain background too much. The proposed lets the width and height little smaller than the bounding box of connected component labeling ($1/1.3$ in our experiment), as the blue dotted line in Fig. 2.8. Also, a tracker is added at a time, because the foreground is more likely to be broken when it appears in the frame at the first time. As mentioned in Chapter 2.10, only the pixel that is foreground is accumulated into the target histogram.

Figure 2.8: Target object box

Chapter 3

Proposed Hardware Architecture

3.1 Architecture Overview

3.1.1 Design Target

We want to design a reconfigurable coprocessor for intelligent surveillance systems. It can be used to accelerate most of the algorithms that are often used in surveillance systems, such as segmentation, tracking, face detection and scoring, object localization, and action recognition. The coprocessor is not only used to accelerate the operation, but also used to verify the algorithms have just been developed with the programmability of the proposed coprocessor. The processors that have been developed by others cannot operate the algorithms efficiently and cost too much. We would like to implement a low cost but efficient design, and we also analysis the operation cycles in both hardware and software to make sure our design is a proper one.

3.1.2 Design Challenge

To design a high performance hardware, the high throughput, high memory, and high bandwidth requirements are the battle necks for many vision algorithms. Large amounts of data are processed in those algorithms, such as pixel data, his-

togram data, frame data, and etc., because the algorithms need to collect large amount of information to complete the computation. The high data and bus bandwidth usage will limit the processing speed of the hardware. It is no use if the hardware can execute in a high speed but the data are not ready for execution.

The programmability is important in our design. The hardware can be used to accelerate many surveillance functions, such as segmentation, tracking, face detection, and action recognition. Heterogeneous data types are used in those algorithms. For example, the results of segmentation are 1 bit, the pixel data are 8 bits, and the bin values store in histogram are 16 bits. Also, the hardware resources need to be reused to reduce the hardware cost.

There are many vision processors and chips with Mega bits on-chip memory and more that 10 million gate count. Some of them only support up to 128×128 frame resolution, and the cost can be even higher if higher frame resolution is required. Those processors are SIMD and use lots of PEs, but if the bus bandwidth is not enough, most of the PEs will idle.Hence, the bus bandwidth should be analyzed and efficiently utilized.

3.1.3 Proposed Design Techniques

The proposed hardware is called Sub-word Level Parallel Streaming Image Signal Processor (SLP-SISP) for intelligent surveillance application. The SLP-SISP uses heterogeneous streaming processing to increase the processing throughput. Streaming interfaces between streams with different data types are designed. SLP-SISP uses a reconfigurable memory array to be the input buffers and output buffers and the memory array can also be the streaming interfaces to buffer the data until they are ready to stream to the next step. The memory array can also be used as the delay line of any data type.

The SLP-SISP also uses sub-word level parallelism to increase the processing throughput, and it can lower the required bandwidth with higher bus utilization. Moreover, the on chip memory can be shared for heterogeneous data streams if

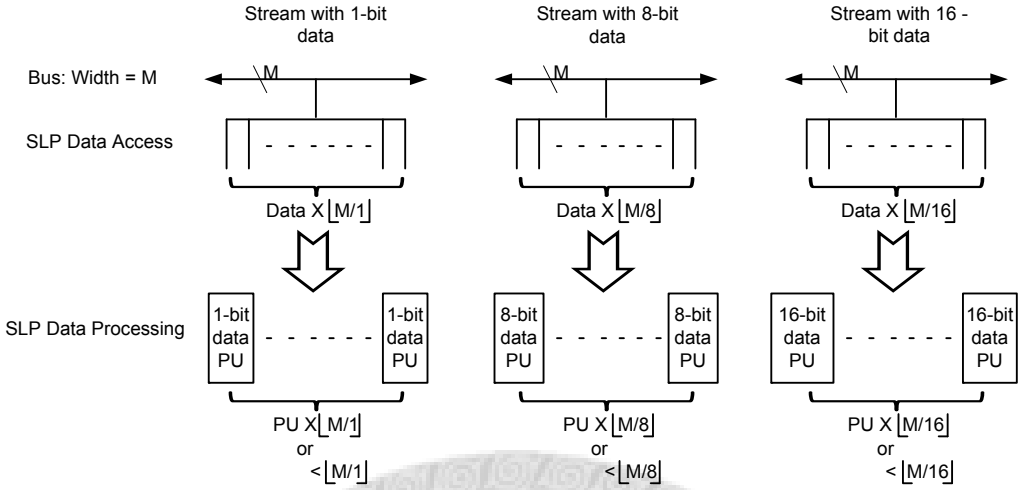


Figure 3.1: Sub-word Level Parallelism data accessing and processing

SLP is applied. The bus that SLP-SISP uses is 64 bits, but the data processed in SLP-SISP are not always 64 bits. The SLP-SISP writes the 64-bit data that contain multiple data into the input buffers in the hardware, and they can be processed simultaneously. For example, in Fig. 3.1, if we want to process some 8-bit data, and the bus width is M, M bits data can still be accessed that contain $\lfloor M/8 \rfloor$ 8-bit data. More than one processing units can be used to process those data at the same time. Also, more than one datum can be accessed when 1-bit or 16-bit data are needed. Sub-word Parallelism can be used to lower the required bandwidth. If the data accessed at the same time is lower than the bus width, the bandwidth is wasted, and it needs more time to access data.

SLP-SISP is a coarse grain reconfigurable vision processor. Many reconfigurable processing units and memory element are design for hardware sharing. It employs many reconfigurable streaming processing elements (RSPEs). The users only have to control the input and output of the RSPEs and select the desired function in the RSPE. The SLP-SISP will complete the computation as they want.

3.1.4 Overview

SLP-SISP is composed by several sets of RSPEs, a main controller, context registers, and it also needs a host processor to co-work, as shown in Fig. 3.2.

The Reconfigurable Interconnections at the mid of Fig. 3.2 (black arrows) are just like shunts of the train rails. The Main Controller controls where the train (data stream) goes, and the clocks of the RSPEs that is not used at that time will be gated to reduce the power consumption. After the route is decided by the Main Controller, the input data stream in and then stream out as an output or are stored in the Reconfigurable Memory Array. This memory array can be used to be the streaming interfaces between streams with different data types, and can be reused. 64-bit data bus, which means the data in and out are also 64 bits, is used. However, the pixels data are 24 bits color (YUV), the segmentation data are 1-bit, the inputs and outputs of CORDIC are 32 bits precision and the bins value of histogram are 16 bits. The Reconfigurable Memory Array is used to be the input buffer and the memory to store those data and use multiplexers to choose some bits of them to stream in our RSPEs. The data burst into the buffers till they are full. A Basic Memory Elements in Reconfigurable Memory Array contains ten 64-bit registers, and every bin value need 16 bits to store, so a set of histogram (8-bit color) need 7 Basic Memory Elements. Three histograms (Y, U, V) are stored for candidate, and three histograms for target, 42 Basic Memory Elements are needed to store the histograms required in our algorithm. Six Basic Memory Elements are used to be the input buffers, so 48 Basic Memory Elements are needed. Histogram Accumulation RSPE is used to accumulate SIFT [19] (Scale-invariant feature transform) and YUV histogram by using the multiplexer. SIFT is an important algorithm in tracking algorithms and computer vision algorithms. It can handle the problem of object rotation and the change or object scale. In [19], the keypoint descriptors are computed through accumulating the histogram of the edge orientations over sub-regions, and those keypoint descriptors can be a feature to make the tracking algorithm better. Histogram Accumulation RSPE can be used

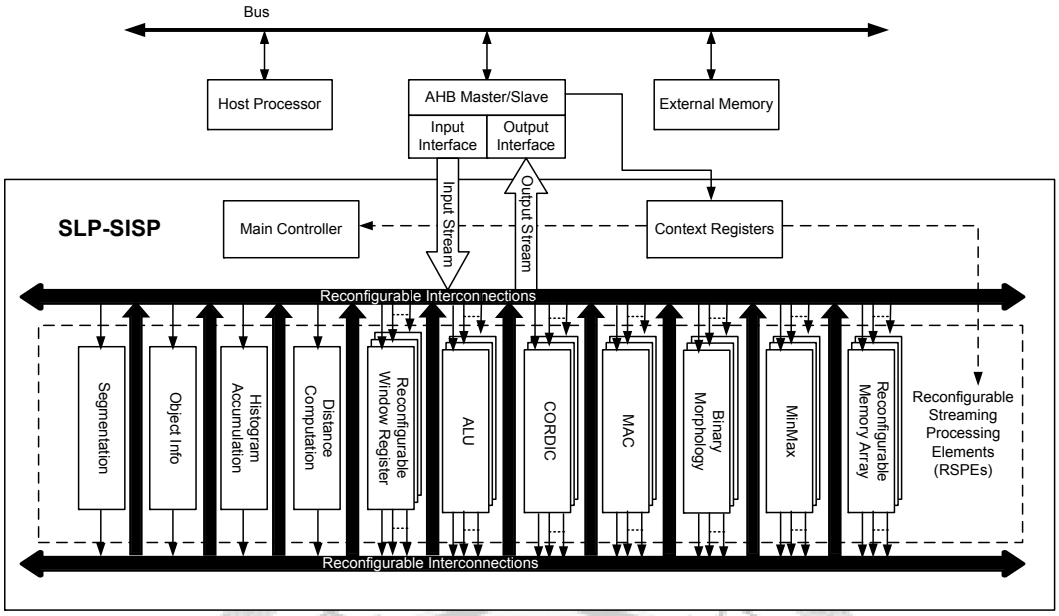


Figure 3.2: Overall Architecture of SLP-SISP

to compute the keypoint descriptor.

CORDICs are used to compute the Δ in Equation2.14 and Δ are written into Reconfigurable Memory Array be the input of Histogram RSPE. The SIFT Converter is used to mapping the positions and the degrees to 256 bits data, and the 256 bits data can be seen as a color with magnitude as Δ.

The users write the data into the Context Registers at the address they want. Those Context Registers are used as the instruction to SLP-SISP and control the whole system. SLP-SISP streams the output data through Output Interface after it completes the computation.

3.1.5 High Memory And Bandwidth Requirements

As mentioned in Chapter 1.3, this kind of media processors or co-processors requires high memory and bus bandwidth. If the design is required to be faster, it has to add lots of memory and PEs. However, they are not added blindly but are added when it is made sure that they are indispensable and reusable. A co-processor is

designed which gets enough speed but is low cost. Reconfigurable Memory Array is designed, and it can be used to be the line buffers of image processing, and can be reused to be the input buffers or histogram bins and so on.

Limitation of bandwidth is also a big problem needed to be solved. As mentioned above, Reconfigurable Memory Array is used to be the input buffers to economize the use of bandwidth. Burst mode is used to move the data into the buffers, and then release the usage of the bus. When SLP-SISP is operating, CPU or other modules can also use the bus.

The bandwidth analysis is described in Table 3.1, and the numbers we estimate here have been considered the bus usage and the data transaction overhead. The computation of SLP-SISP are analyzed, and the cycles needed in a second when the tracking algorithm is performed is calculated. The bandwidth is the bottleneck.Bandwidth 1 and Bandwidth 2 are two kind of histogram accumulation: Bandwidth 1 accumulates histogram without background color removal and Bandwidth 2 accumulates histogram with background color removal. That is, Bandwidth 2 uses the $Mask(x,y)$ in Eq. 2.38 and $choose_y(x,y)$, $choose_u(x,y)$, $choose_v(x,y)$ in Eq. 2.16, and SLP-SISP only accumulates the pixels which are foreground. ARM Developer Suite [45] is used to estimate the number of cycles the proposed algorithm has to use, and ARM926EJ-s is used to be the SLP-SISP's host CPU. The CPU loading is shown in Table 3.2. The number of the object is assumed to be 10 and the number of particles on each object is 100, the size of particles is 50×50, here. Morphology is the morphological operations and used to de-noise the result of segmentation to make the result better. Connected Component Labeling and Object Info are used to calculate the size and position of objects. When spreading particles, host CPU is used to calculate the position and the size of the particles that have to be spread, and write the information into the hardware. After get the result of histogram comparison from the hardware, host CPU calculates HDist from Eq. 2.35. The calculation takes lots of cycles to compute here, however, the users can simplify it and design a new equation here.

After that, in Particle Weight Computation, the $Count_t^k$ in Eq. 2.37 is calculated by hardware, and the whole equation is computed host CPU. In Object State Estimation, CPU uses the weight and the parameters of the particles to estimate the objects' current states by Eq. 2.6.

SLP-SISP can be used to track 10 objects and 100 particles for each object (50 particles for position and 50 particles for size), when 125 MHz clock is used, with background color removal.

Table 3.1: Bandwidth analysis of tracking hardware (8MB/s)

Steps	Bandwidth 1	Bandwidth 2
Segmentation	29.7	29.7
Morphology	0.68	0.68
Connected Component Labeling	7.38	7.38
Object Info	1.37	1.37
Add New Object	0.00	0.00
Accumulate the New Objects' Color Histogram	0.02	0.03
Spread Particles	0.00	0.00
Histogram Accumulation and Comparison	17.55	41.18
Particle Weight Computation	1.58	1.58
Particle Resampling	0.00	0.00
Object State Estimation	0.00	0.00
Model Update	2.91	2.96
Total	61.19	84.88

3.2 Reconfigurable Memory Array

As mentioned above, 48 Basic Memory Elements which are 10×64-bit in Reconfigurable Memory Array are used. The Reconfigurable Memory Array is used to

Table 3.2: Host CPU Loading (MHz)

Steps	Host CPU Loading
Segmentation	0.00
Morphology	0.00
Connected Component Labeling	0.00
Object Info	0.00
Add New Object	0.00
Accumulate the New Objects' Color Histogram	0.00
Spread Particles	17.42
Histogram Accumulation and Comparison	0.0
Particle Weight Computation	179.36
Particle Resampling	1.20
Object State Estimation	17.62
Model Update	17.94
Total	233.54

be the line buffers, histogram bin data registers, temporal data registers of distance computation, input buffers, output buffers, and so on. Moreover, the data of every register can be accessed directly by the user, that is, those registers can be used to test the functional of our chip, just like the scan chains. SLP-SISP can write or read the data with the corresponding address. The Reconfigurable Memory Array is reusable and reconfigurable. The user can use Reconfigurable Interconnections which are controlled by Main Controller to control the input and output of the memory array. The Reconfigurable Interconnection is just like a switch to connect the Reconfigurable Memory Array to the RSPEs as we want.

Because our design target is a 640×480 frame, 640 bits in a Basic Memory Element is used. It can be the line buffers of the result of segmentation, and a line of 1-bit mask data can be stored in a Basic Memory Elements. By doing so, the post-segmentation work can be implemented easier. Dilation, Erosion, and some other 1-bit window operation must use line buffers to reorder the data, because they all need the data of previous line and next line. With line buffers, SLP-SISP can do the window operations efficiently with parallel architectures, which is 64 SLP for 1-bit window operation.

It is needed to have $16\ bits \times 256\ bins = 7$ Basic Memory Elements to accumulate one color of histogram, $7 sets \times 3(YUV) \times 2(target\ and\ candidate) =$ 42 Basic Memory Elements to compute the distance of histogram, and 6 Basic Memory Elements to be the input buffers. As a result, 48 Basic Memory Elements are needed totally.

The Basic Memory Element is two ports and can be cascaded together to be a long line buffer as shown in Fig. 3.3. In the figure, the A and B are the two ports, and 8 Basic Memory Elements are cascaded to be a 640×8-bit line buffer. More Basic Memory Elements can be cascaded for larger data size, such as 16-bit.

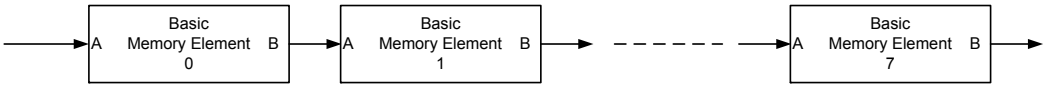


Figure 3.3: Reconfigurable Memory Array as line buffer for 8-bit data

3.3 Histogram Accumulation RSPE

This RSPE can be used to accumulate histogram, count white pixels of the result of segmentation, and accumulate the SIFT histogram. The 64-bit data stream into the data buffers until they are full. There is a position counter to count where the pixel is in the image to compute the Δ in Eq. 2.14. The computation of Δ can be done by CORDIC and it will be described in Chapter 2.5.

As shown in Fig. 3.4, after the buffers are full, the data get into the Histogram Accumulation unit. The gray parts in the figure are all Reconfigurable Memory Array. 2 pixels of data operate simultaneously. First we compare these two pixel colors are the same or not, and we watch the flags of these two colors are set or not, that is, the bins are written before or not. If the flags are set, we will read the corresponding data from the Reconfigurable Memory Array, and if not, we will use 0 to be the read data. If the two pixel colors are the same, the two data read from Data Buffer are added to the read data and then be written to the reconfigurable Memory Array. If the two pixel colors are different, the two are added and be written out separately. By this design, we can operate two pixels at the same time. Also, the design of flag will reduce the frequency of reading data, and this will reduce some power.

The MASK Buffer is used to buffer the data from the result of segmentation, and the stored data is used to calculate $Mask(x,y)$ in Eq. 2.38. The 1-bit data is used to choose whether the pixel will be accumulated into the histogram or not.

The histogram bins are 16-bit precision but the word length of the Reconfigurable memory Array is 64-bit. Histogram Memory Converter is used to make a mapping from bin indexes to the memory address, and a mask is also used to

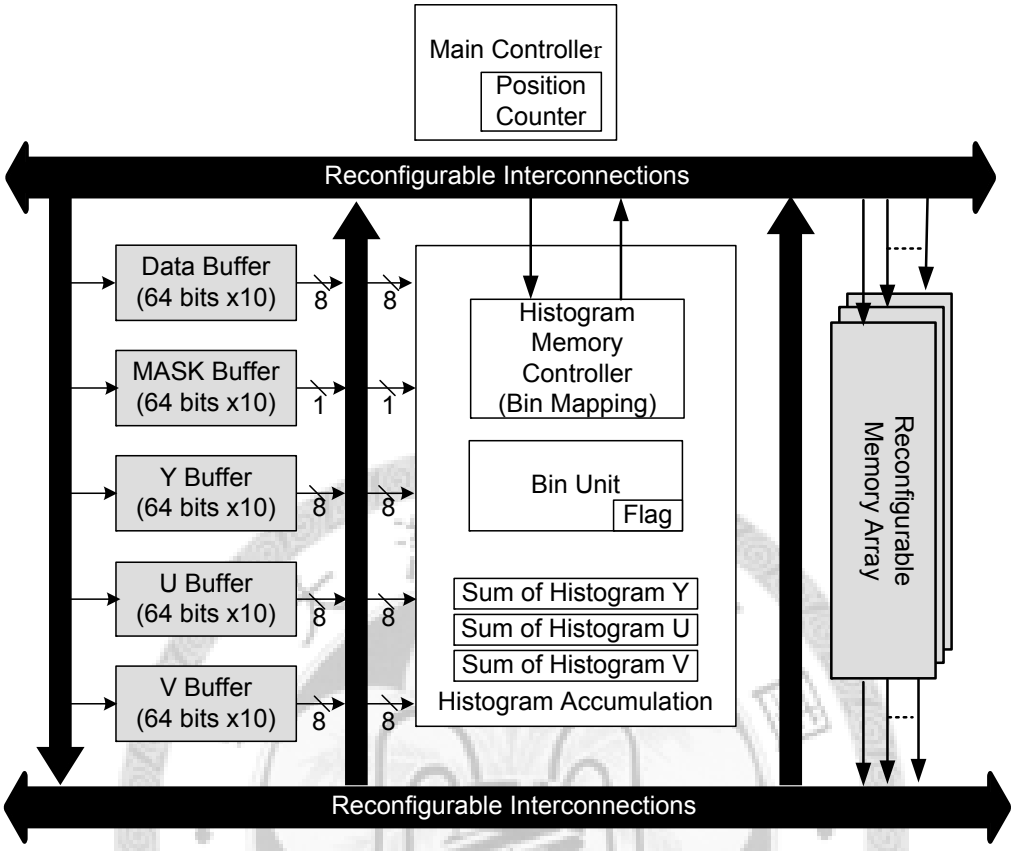


Figure 3.4: Histogram Accumulation RSPE

choose which 16 bits will be read out or write in, and others are stay unchanged. A Basic Memory Element is 64 bits with ten addresses, and a color histogram for a color channel needs 7 Basic Memory Elements to store its data.

The total sum of histogram Y, U, and V is accumulated.These three values are calculated by three adders and add the input from CORDIC/SIFT Data Buffer, and 20-bit register is used to store each value. If centroid emphasizing mode is used to accumulate the histogram, 16-bit bin and 20-bit total quantity can be used to accumulate the particle size about 100×100 with 80×80 region of the same color. If the objects are too big, the down-sampling of the video frame can be applied in the first place.

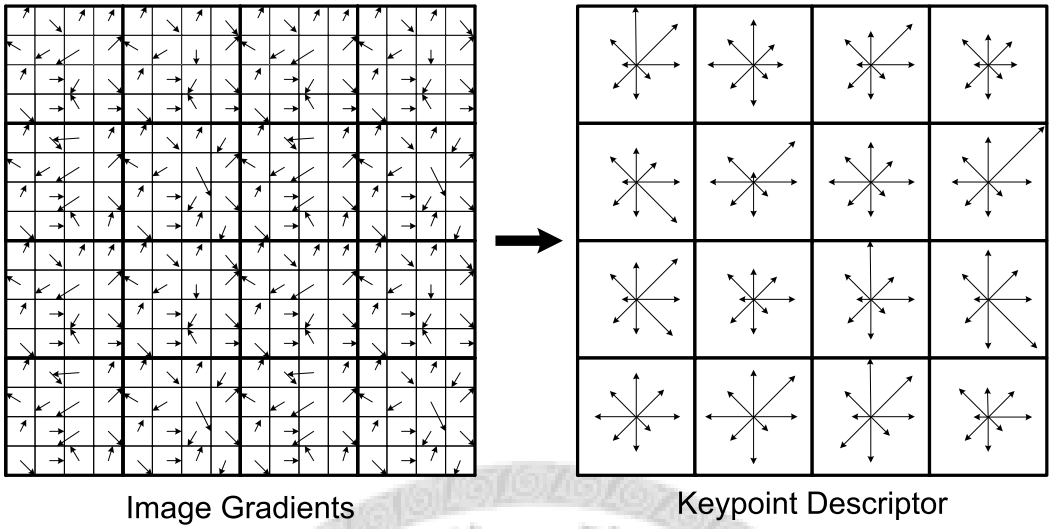


Figure 3.5: Keypoint Descriptor

3.3.1 SIFT Converter

After the image gradients is gotten from the difference-of-Gaussian functions, the image gradients is used to make the keypoint descriptor. A 16×16 image gradient array is used to compute a 4×4 descriptor array, in which a 4×4 image gradient array is used to compute one descriptor, as shown in Fig. 3.5. The orientation is represented by 3 bits and the magnitude is represented by 8 bits. This converter is used to convert the orientations (positions and degrees) to the histogram bins ($0 \sim 127$). Four bits are used to represent x-coordinate and y-coordinate respectively in Fig. 3.5. The least significant two bits are the coordinates in the 4×4 blocks, and the other two bits are the coordinates of blocks in 4×4 sub-regions. The degree of orientation can be present by the 3-bit data, as shown in Table 3.3. In this converter, the converter normalize the degree to $0^\circ \sim 360^\circ$ first. Then Table 3.3 is used to map them to the 3-bit data. After all, the 3-bit data and the most significant 2 bits of x-coordinate and y-coordinate are combined together to be the bin address to Histogram Accumulation RSPE.

Table 3.3: The degree of orientation

Degree of Orientation (°)	3-bit Data
0∼22.5 or 337.5∼360	0
22.5∼67.5	1
67.5∼112.5	2
112.54∼157.5	3
157.5∼202.5	4
202.5∼247.5	5
247.5∼292.5	6
292.5∼337.5	7

3.4 CORDIC

Coordinate rotation digital computer (CORDIC) is a simple and efficient algorithm to calculate rotation, the distance between two points, the angle between two lines and so on. It is easy to implement in hardware as Fig. 3.6.

Five iterations of CORDIC is unfolded in the CORDIC design. The hardware is 32-bit fixed-point precision and the digital point is between 16^{th} and 17^{th} bits. The precision of the answer can be about 98%.The control in the middle of Fig. 3.6 is used to control the behavior of the multiplexers. Every multiplexer is affected by the output of the previous iteration, or said, the previous pipeline. Two modes can be chosen with the Choose port. CORDIC can be used to calculate the angel between the vector (X_in, Y_in) and the positive axis X, or calculate the length between the point (X_in, Y_in) and the original point $(0,0)$. Because the clock cycle is not enough, we cut the design into five pipelines. They are not folded together, because if they are folded, 5 cycles for each operation are needed to be spent. This architecture is based on the following formula [46]: Rotation mode:

$$\begin{aligned} x_{i+1} &= x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + x_i \cdot d_i \cdot 2^{-i} \\ \theta_{i+1} &= \theta_i - d_i \cdot tan^{-1}(2^{-i}) \end{aligned} \tag{3.1}$$

where

$$d_i = \begin{cases} -1, & \theta_i < 0 \\ +1, & else \end{cases} \tag{3.2}$$

The result is gotten as

$$\begin{cases} x_n = A_n[x_0 cos\theta_0 - y_0 sin\theta_0] \\ y_n = A_n[y_0 cos\theta_0 + x_0 sin\theta_0] \\ \theta_n = 0 \\ A_n = \prod_{i=0}^{n} \sqrt{1 + 2^{-2i}} \end{cases} \tag{3.3}$$

Vectoring mode:

$$\begin{aligned} x_{i+1} &= x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + x_i \cdot d_i \cdot 2^{-i} \\ \theta_{i+1} &= \theta_i - d_i \cdot tan^{-1}(2^{-i}) \end{aligned} \tag{3.4}$$

where

$$d_i = \begin{cases} +1, & y_i < 0 \\ -1, & else \end{cases} \tag{3.5}$$

The result is gotten as

$$\begin{cases} x_n = A_n \sqrt{x_0^2 + y_0^2} \\ y_n = 0 \\ \theta_n = \theta_0 + tan^{-1}(\frac{y_0}{x_0}) \\ A_n = \prod_{i=0}^{n} \sqrt{1 + 2^{-2i}} \end{cases} \tag{3.6}$$

The multipliers in the right side of Fig. 3.6 are used to divide A_n (about 1.6457) in Eq. 3.1 and 3.4, because it can be changed to be multiplied by 0.607648. The constant in the figure is the value of $tan^{-1}(2^{-i})$ in Eq. 3.1 and Eq. 3.4. Because

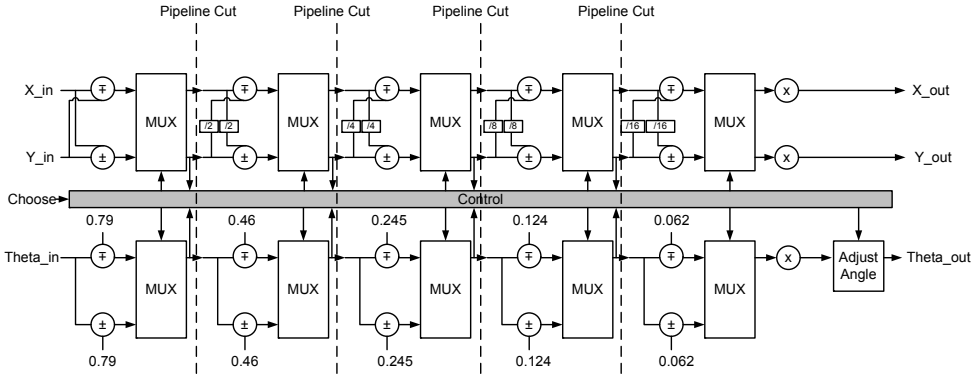


Figure 3.6: CORDIC RSPE

the number of iteration is set to 5, those variables can be constants, and the computation can be simplified. The Adjust Angle is used to transfer the angle (θ_n) from $-\pi/2 \sim \pi/2$ to $-\pi \sim \pi$ by the sign of x-coordinate and y-coordinate.

3.5 Distance Compute RSPE

Diffusion Distance mentioned in Chapter 2.7.1 is used to compare two histograms. As shown in Fig. 3.7, the architecture is just like the architecture of Histogram Accumulation RSPE. Both of them need a bin mapping, which is in the middle of Fig. 3.7 and is called Distance Memory Controller (DMC), to map the bin index to the address of Reconfigurable Memory Array. Two bin data from candidate histogram and two bin data from target histogram are read in at the same time. This part of computation will be discussed later. A Distance Compute RSPE can compare two histograms, so three Distance Compute RSPEs are used to compare the YUV histogram between the candidate and the target.

As shown in Fig. 3.8, it is the architecture of the Distance Compute RSPE. At the first iteration, two candidate histogram bin data and two target histogram bin data are read in at the same time, and the candidate histogram bin data are subtracted from the target histogram bin data. The multiplications here are used to normalize the histograms. The data are distributed into odd-bin-number and

even-bin-number, because down-sample is needed in the Eq. 2.32. The middle of the Fig. 3.8 is used to compute the convolution in the Eq. 2.32. The window size of ϕ as (-3, 3), and t as 0.8 is chosen. The coefficients of ϕ are multiplied and the results are stored in the shift-registers as shown in the middle of Fig. 3.8. The brown registers are used to calculate ϕ convolution of the odd-bin-number and the blue ones are used to calculate ϕ convolution of the even-bin-number. It can be explained by the description below. If the shift-register is unfolded as the sketch map in Fig. 3.9. Four rows mean 4 different multipliers (coefficient) which can be get by use Eq. 2.33, which t = 0.8, i = 0, 1, 2, 3. The RSPE does not need 7 rows to compute ϕ because the same ϕ can be gotten when positive i and negative i are used. It can be said that Fig. 3.9 is the window to do the window operation.

After the 14 values from Fig. 3.9 are gotten, they are added to be the value of convolution in the Eq. 2.32, respectively. Add the both ϕ and the absolute of the difference of the 4 candidate histogram bin data, which have been calculated before, to be the distance of the first iteration, and do it again and again until all bin values are calculated. The two ϕ are added and divided by 2 to be the down-sample value and written out to the Reconfigurable Memory Array. After all the bin values are calculated, the first iteration is finished. Then the down-sample value are read back at the second iteration. The second iteration is similar to the first iteration, but when the distance is computed, it is not needed to add the absolute of the difference of the 4 candidate histogram bin data. After five iterations, the result of comparison can be gotten.

3.6 Object Info RPSE

After perform connected component labeling, it is needed to know the bounding box of the object. This RSPE is used to calculate the bounding box of all objects with given object ID, and it can calculate ten objects at the same time. The input data are also 64 bits, that is, 8 pixels' data stream in simultaneously. Two counters

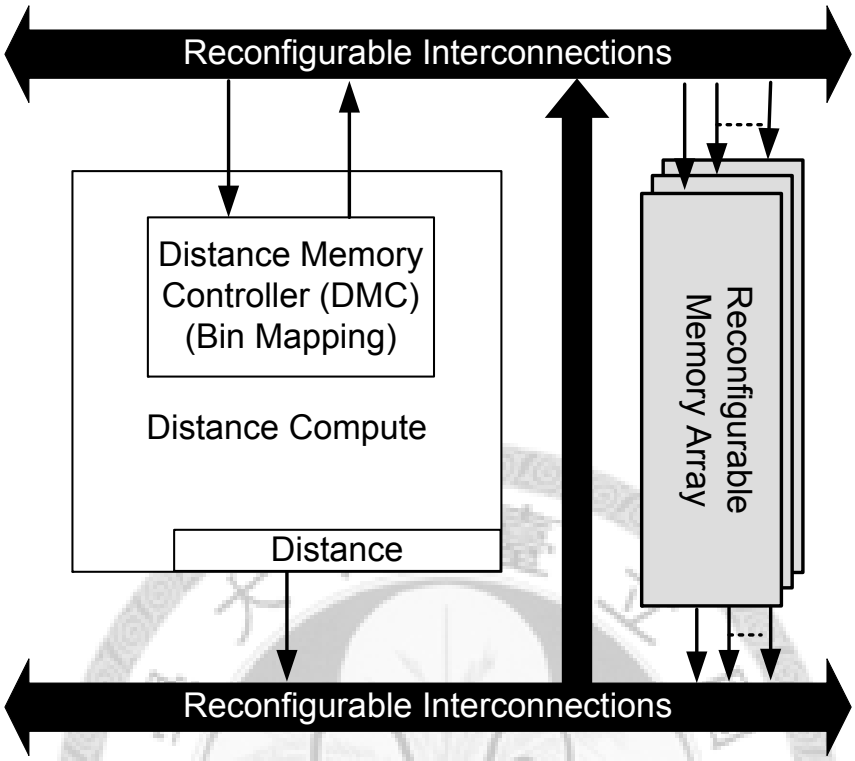


Figure 3.7: Distance Compute RSPE

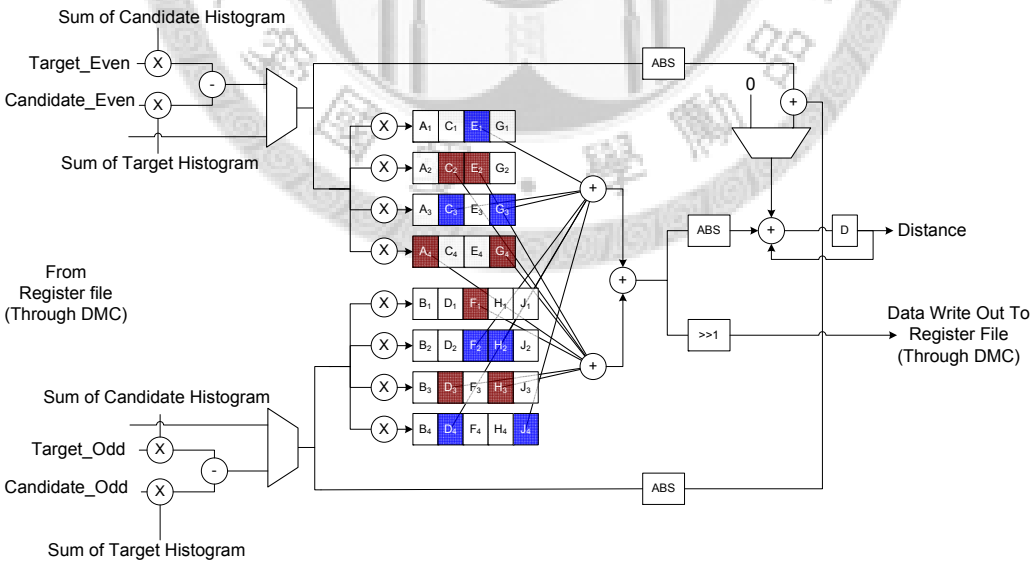


Figure 3.8: The Inside Of Distance Compute RSPE

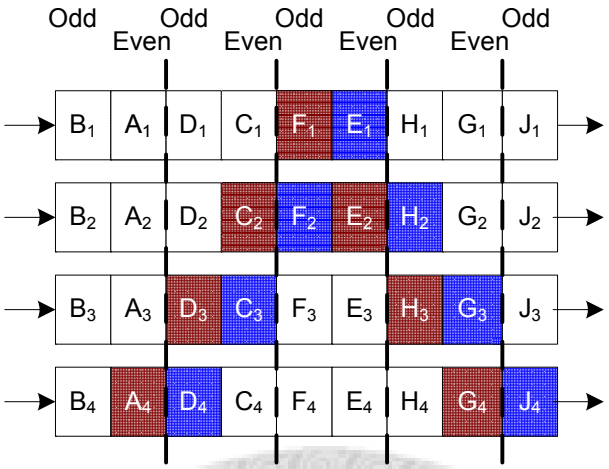


Figure 3.9: The sketch map of the coefficient of Distance Compute RSPE

(Counter-i and Counter-j) are used to count the pixel coordinate X and Y. Add 1 on Counter-i every cycle until Counter-i$\geq$ 80, and Counter-i will become 0 and Counter-j will be added 1 on. Because there are 8 small PEs to count the bounding box in horizontal direction and the frame size is 640×480, the Counter-i does not need to count to 640 but count to $640 \div 8 = 80$. The PEs are shown in the middle of Fig. 3.10. The principle of Object Info RSPE is that the RSPE gather the min and max value of the x-coordinate and y-coordinate of the object of corresponding ID. The RSPEs are used to output the coordinate if the coordinate is smaller than the min value or bigger than the max value they have stored.

The right hand part of Fig. 3.10 is used to find out which PE is the min value from. The index in the red rectangle is used to indicate the value of the PEs' output is valid or not. If the min x-coordinate of the object is needed to be calculated, the following steps will be done. First, the RSPE checks whether there is any valid output in top 4 PEs or not, and that is what the first multiplexer doing. If those there is not valid output in top 4 PEs, the valid output is in bottom 4 PEs. Then the RSPE checks whether the outputs of top two PEs, which are in the 4 PEs, which have valid output and we have chosen in the first step, are valid or not. At the

third step, the RSPE finds out which the PE of the two PE have been chosen in the second step have the valid output, and that output is the min value. For example, in Fig. 3.11, if the white points mean the location of the object and black points are the background. First, the top 4 PEs are checked, and there is no valid output. Then the top 2 PEs (i+4, i+5) in the bottom 4 PEs (i+4, i+5, i+6, i+7) are checked, and there is valid output. At final, the top PE (i+4) in those 2 PEs (i+4, i+5) are checked, and the top PE's output (i+4) is not valid, and (i+5) is the min value.

If the max x-coordinate of the object are needed to be calculated, the steps are a little different. For example, in Fig. 3.12, first the RSPE check the bottom 4 PEs, and there are valid outputs. Then the RSPE check the bottom 2 PEs (i+6, i+7) in the bottom 4 PEs (i+4, i+5, i+6, i+7), and there is no valid output. At final, the RSPE check the bottom PE (i+5) in those 2 PEs (i+4, i+5), and the bottom PE's output (i+5) is not valid, and the RSPE can get that (i+4) is the max value.

The third multiplexer is used to handle that if there is no valid output of 8 PEs, the min value stay unchanged. The 9th PE is used to calculate the min or the max of y-coordinate. Because we only handle 1 y value at a time, this part of architecture is easy. Every object need 2 Object Info RSPEs, and one is for max and the other is for min. Ten objects are need to be calculated at the same time, so 20 Object Info RSPEs are needed. However, the Object Info RSPEs are small and low cost.

3.7 Reconfigurable Window Register RSPE

It is used to support the window operation. It is a 64-bit shift register as shown in Fig. 3.13. The Input comes in from the right side of the figure. The data shift from right to left, and the light blue part of data shift to the blue part of data in the next cycle. The output is 88-bit data. If 8-bit data are used, three Reconfigurable Window Register RSPEs can be used to perform eight 3×3 window operations simultaneously, and 3×10 data are needed to do that. For example, the center can

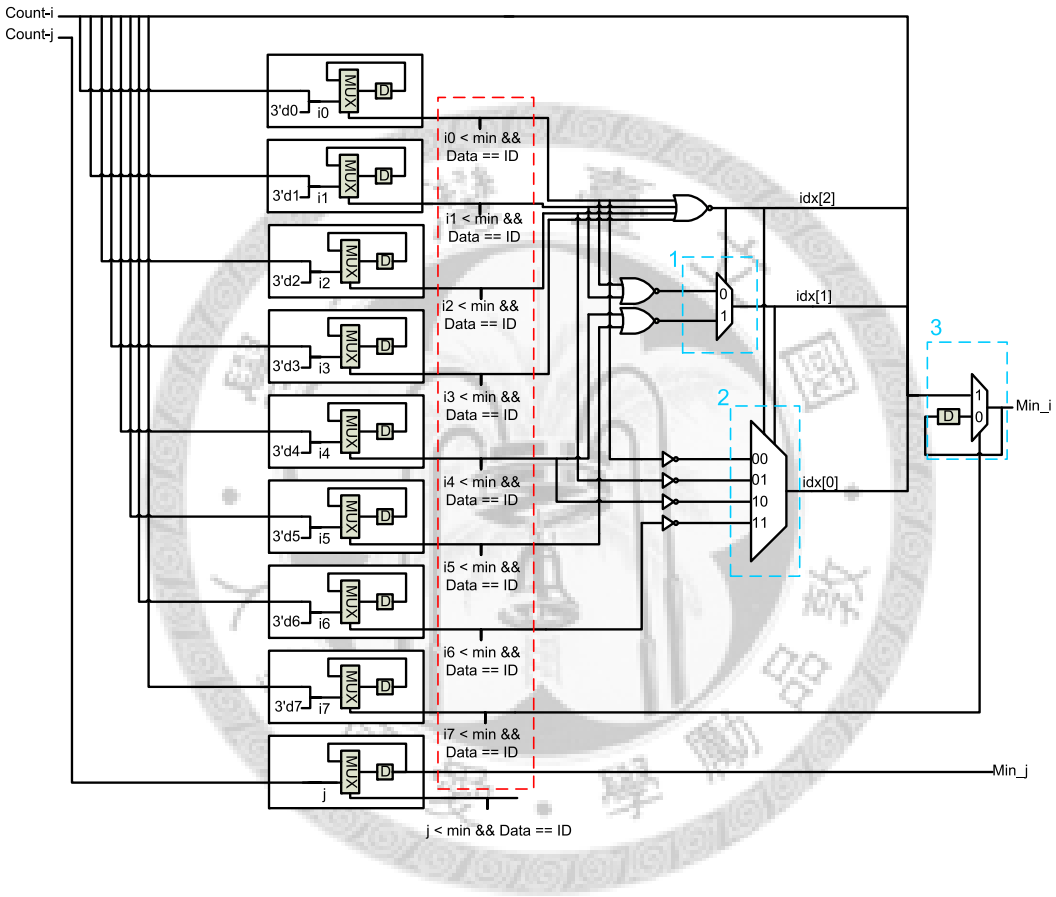


Figure 3.10: Object Info RSPE

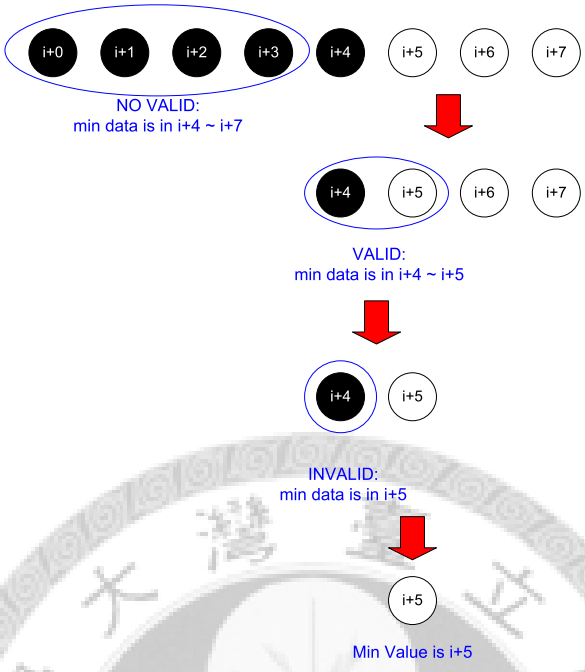


Figure 3.11: An example of Object Info RSPE

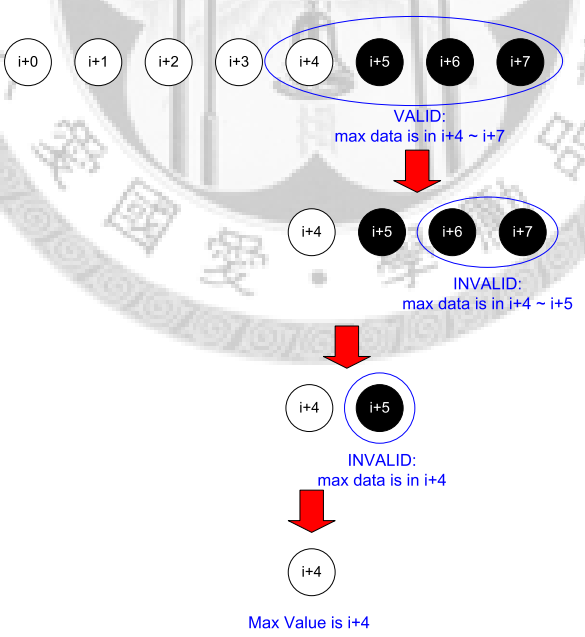


Figure 3.12: Another example of Object Info RSPE

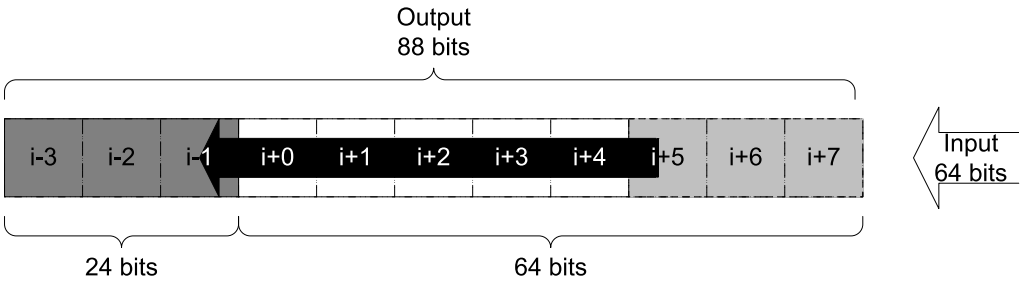


Figure 3.13: Reconfigurable Window Register RSPE

be $i-1, i+0, i+1, \cdots, i+7$, and we have the 10 data, and use them to perform the window operations. Not only eight 3×3 window operations can be performed, eight 5×5 window operations can also be performed by using 5 Reconfigurable Window Register RSPEs. This RSPE can also be used to process 1-bit data, such as performing the morphological operation on the result of segmentation.

3.8 ALU RSPE

The inputs and the outputs of the ALU RSPE are 8 bits, and there are 10 operations that can be chosen. And, or, addition, subtraction, subtraction and divided by 2, absolution of difference of subtraction, multiplication, and divided by 256, minimum, and maximum. The inputs which are from the inputs or from the context registers can also be chosen. The architecture of ALU RSPE is shown in Fig. 3.14. It can be used to calculate Connected Component Labeling.

3.8.1 Connected Component Labeling

This is used to implement the algorithm mentioned in Chapter 2.4. It is not a RSPE, and just uses the ALU RSPEs to support it in our design. The architecture is shown in Fig. 3.15 and the design of PE is shown in Fig. 3.16. This design needs to operate for three iterations. At the first iteration and the third iteration, the input

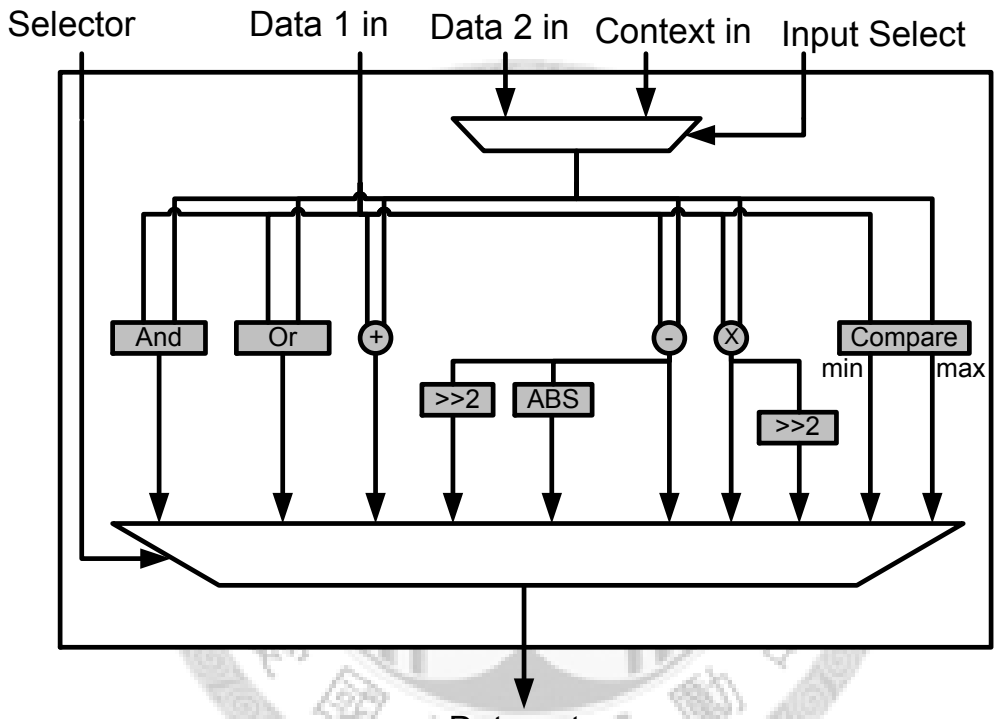


Figure 3.14: ALU RSPE

data are top-down and the data are bottom-up at the second iteration. Only two PEs are used because the targeted clock period is too short to add other PEs, and cutting pipeline is not suitable for the design. The D flip-flops are used to store the data out of PE1 and be inputs to the PE0 at the next cycle, because every PE needs three directions of data: top, left and itself in top-down and bottom, right and itself in bottom-up. The D flip-flops store the data and make the data be the inputs to PE0' s left input (or right input) at the next cycle. In Fig. 3.16, I means the result of segmentation, that is, 1 means foreground and 0 means background. D means the connected component label and it is a 7-bit data. First we stick I and D together to be a 8-bit data. The RSPE compares the value of the 8-bit data from top (or bottom) and from left (or right), and Dtemp stores the value of the bigger one. The PE_Count at the left-down corner is used to generate new labels. If the current pixel is foreground and the top (or bottom) and left (or right) pixel is background, the PE_Count will be subtracted two from. Then we compare the value of Dtemp and the value of the target pixel, and output the bigger one. It always subtracts two from PE_Count because there are two PEs and the values of PE_Count of the PEs cannot be the same.

3.9 MinMax RSPE

It is used to get the minimum and the maximum value of 27 data at the same time and also know which data are the minimum or the maximum. The input data are 8 bits, and we stick the data ID ($0 \sim 26$) on them. After the data flow through the comparator tree in the middle of Fig. 3.18, the RSPE can get the minimum and the maximum with the data ID.

The comparator shown in Fig. 3.17 is the key of the comparator tree. It only compares the value of the least significant 8 bits, but the outputs are still 13 bits, with the most significant 5 bits being the IDs of the data.

MinMax RSPE can be used to calculate the 2-D morphological operation,

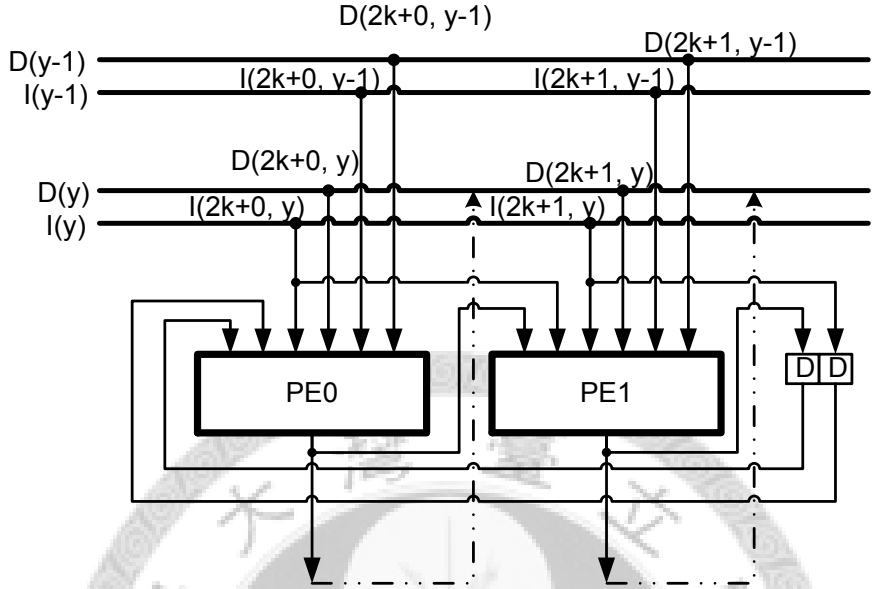


Figure 3.15: Connected Component Labeling

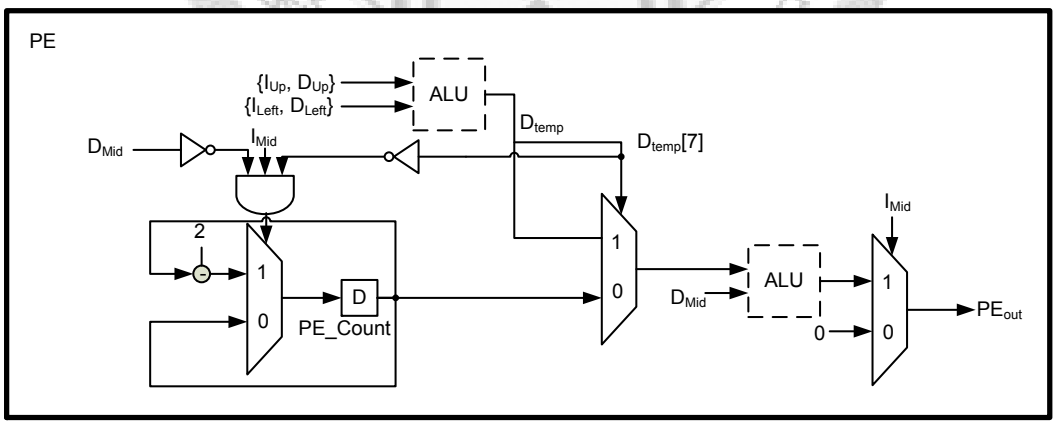


Figure 3.16: The PE of Connected Component Labeling

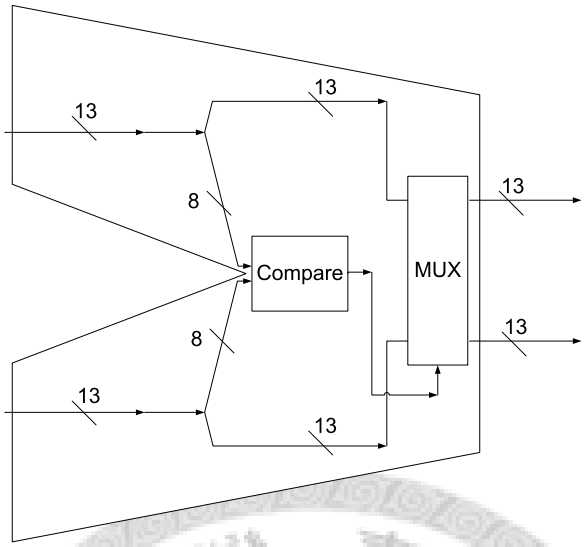


Figure 3.17: The comparator in MinMax RSPE

such as erosion and dilation. In segmentation, MinMax RSPE is used to find the minimum weight for background model update or replace. When the SIFT algorithm is used, it is also needed to find the key points which are the maximum and minimum of the difference-of-Gaussian images and be detected by comparing the pixels to their 26 neighbors in $3 \times 3 \times 3$ regions at the current and adjacent scales [19].

3.10 Segmentation RSPE

As mentioned previously, four layers of background are used to do the object segmentation, and these four layer should be transferred into the design. First, it uses MinMax RSPE to choose the background which has the lowest weight, and uses the Switch in the middle of Fig. 3.19 to switch it to the first layer of background. This step is used to keep the weight of the first layer of background being always the lowest one. Then the RSPE uses the SegCore to compare the difference between the current pixel's value and all background values, and update

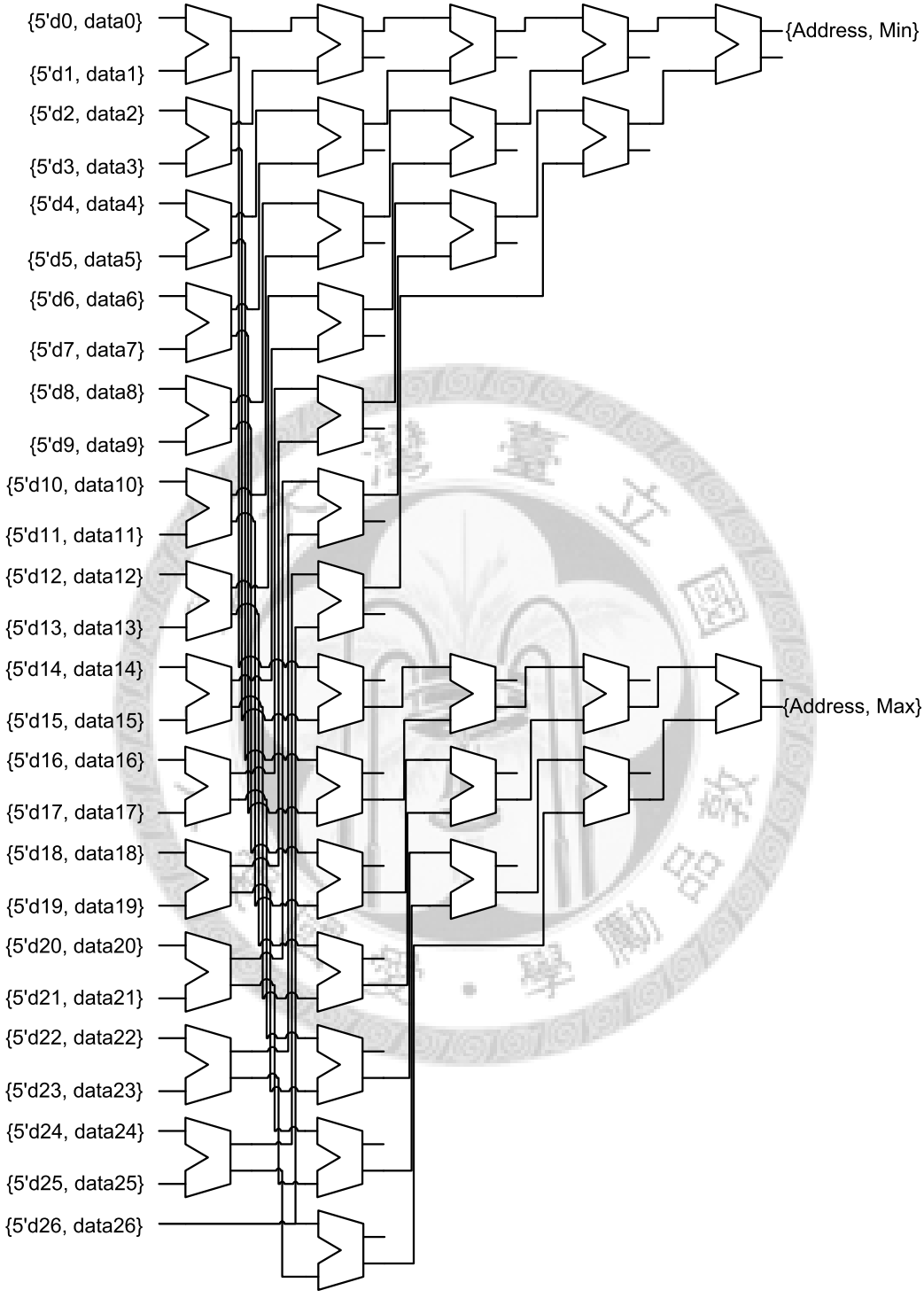


Figure 3.18: MinMax RSPE

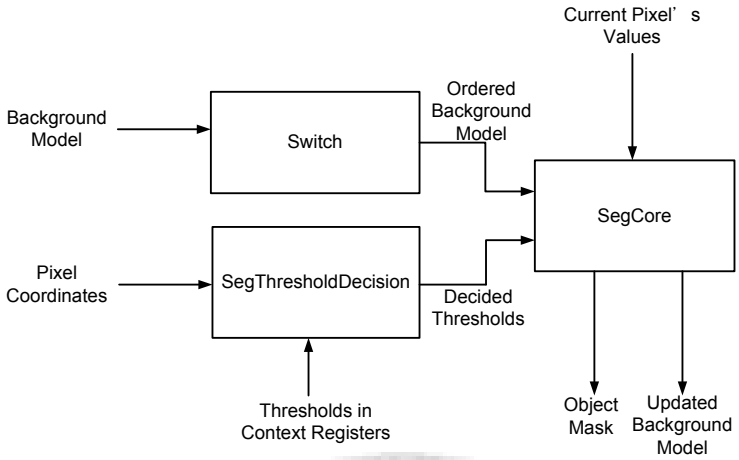


Figure 3.19: Segmentation RSPE

or replace the background model. The Object Mask is made in this step. The SegThresholdDecision is used to decide the thresholds used in SegCore. The users can use the context registers to decide the thresholds and whether the background should be updated or not.

The input buffers in front of the Segmentation RSPE are also used, so burst mode can be used to move the data into the RSPE. One pixel is processed by Segmentation RSPE at a time.

Chapter 4

Experiment Result

4.1 Algorithm Experiment Result and Analysis

50 particles are used for each object in the experiment result. 25 particles are used to estimate the position of target object and 25 particles are used to estimate the size of target object.

First the result with Bhattacharyya Distance, Diffusion Distance, and Earth Mover's Distance in color histogram comparison are compared. The results is shown in Fig. 4.1, Fig. 4.2, Table 4.1, and Table 4.2. In these results, we do not combine the result of segmentation to determine the weights of particles.

A sequence is shown in Fig. 4.1. A battery roll through a sun light region and into a dark region. As start in (A), the results of four algorithms are the same. When the lighting condition starts to change, the results of 1D Bhattacharyya Distance and 3D Bhattacharyya Distance get wrong in (B). Although new trackers on both Bhattacharyya Distance are added, the result of 3D Bhattacharyya Distance gets worse in (C). The color in Y channel changes a lot but in U and V channel are not change so much. In this situation, 1D Bhattacharyya Distance is a little bit better than 3D Bhattacharyya Distance. After the battery roll through the light region, all results but the result of Diffusion Distance get wrong in (E). The trackers all stop at the light region, because the light condition changes rapidly.

(F) shows that if the result be estimated is right in previous frames, the result can still be right although there are another object with same color (another battery) in the frame, because the position of the object location is estimated to spread the particles first.

Another sequence is shown in Fig. 4.2. It is just a sequence with a little shadow at the corner of the frame. However, a low quality camera is used to capture the sequence, and the color in the frame is flickered all the time. The results of 3D Bhattacharyya Distance and 1D Bhattacharyya Distance are not stable, and can be shown in (B) and (C). The result of 1D Bhattacharyya Distance gets wrong in (E), but the others are still good. The results of Diffusion Distance and Earth Mover's Distance are all good in this sequence.

In Table 4.1, some quantitative results are given. The first to the third test sequence are Fig. 2.6 , Fig. 4.1, and Fig. 4.2. The fourth sequence is that a pen roll on a table with flickering light condition. The fifth sequence is that a man walk in a dark hall. In this sequence, the data in channel U and V are useless, so the result of 3D Bhattacharyya Distance is not good. If the tracker and the object are overlap, the frame is a correct frame. The number of correct frame divided by the number of total frame equals the number in the table. The higher the value is, the algorithms is better.

Also we can calculate the position error here, as shown in Table 4.2. The result of tracking is compared with the ground truth, and the distance is calculated.

The importance of combining segmentation result in particle weight determination and α in Eq. 2.37 can be shown in the sequence if the foreground color is similar to the background color. The performance may decrease, and Fig. 4.3 is an example for this. $\alpha = 0$ and $\alpha = 0.4$ are chosen to be compared. This sequence is that a guy in a white T-shirt and black pants walk through a white wall with a black window on it.

In (A) to (E), the result is almost the same good. In (F) the tracker in $\alpha = 0$ starts to be trapped by the black window, and it causes an error in (G). The tracker

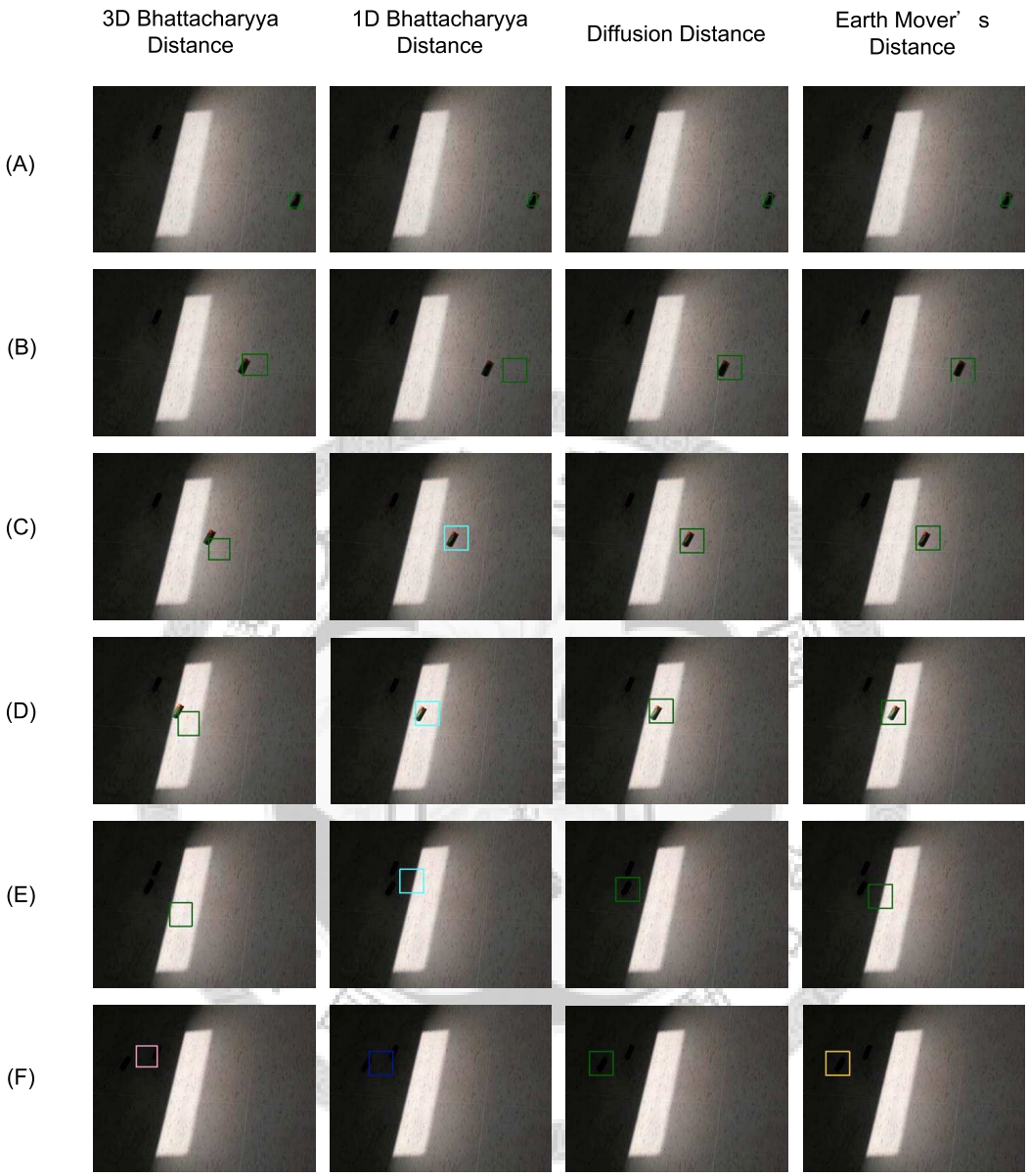


Figure 4.1: Comparison of three algorithms. The first column is 3D Bhattacharyya Distance, the second column is 1D Bhattacharyya Distance, the third column is Diffusion Distance, and the fourth column is Earth Mover's Distance. (A) frame no.1 (B) frame no.18 (C) frame no.35 (D) frame no.52 (E) frame no.69 (F) frame no.86

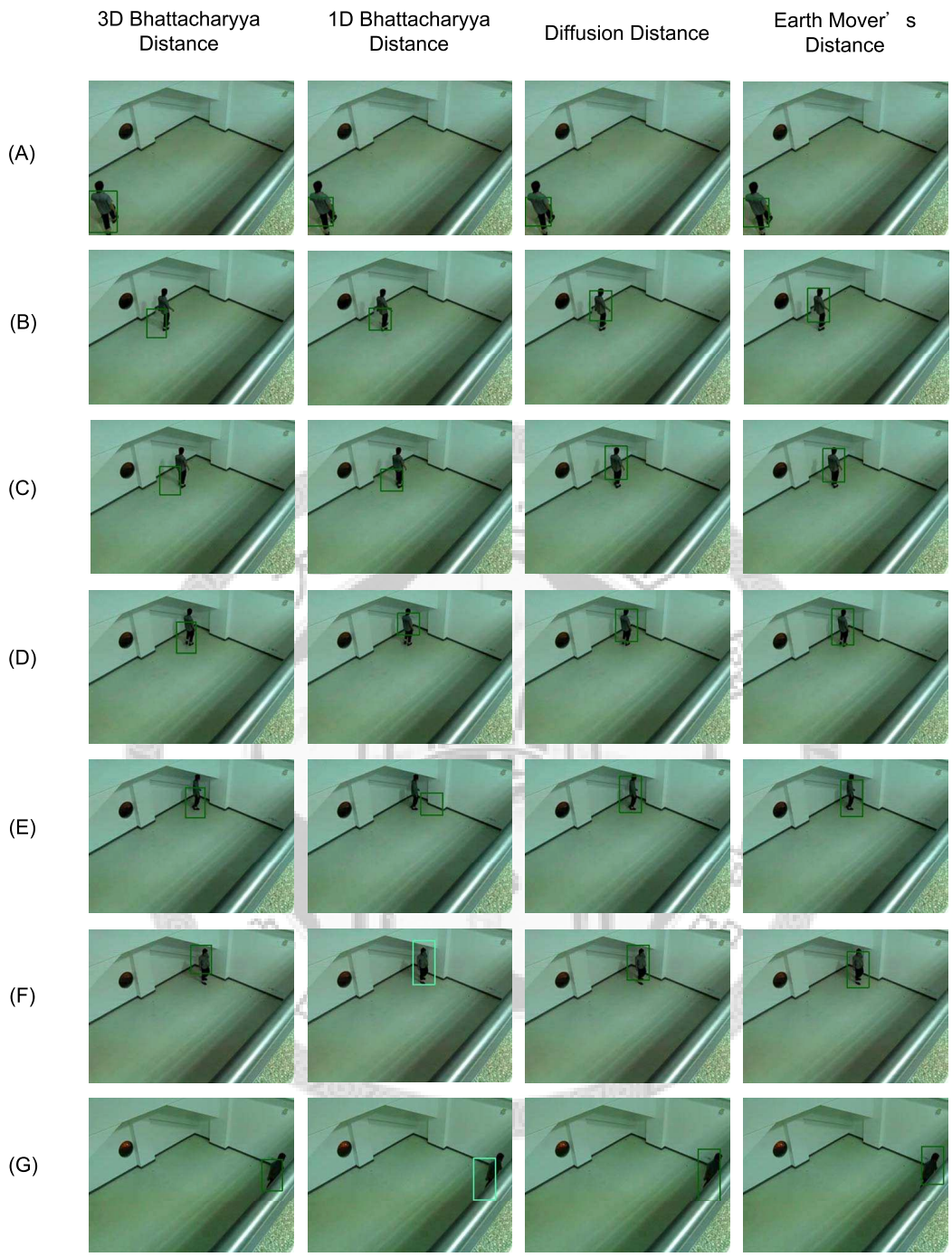


Figure 4.2: Comparison of three algorithms. The first column is 3D Bhattacharyya Distance, the second column is 1D Bhattacharyya Distance, the third column is Diffusion Distance, and the fourth column is Earth Mover's Distance. (A) frame no.51 (B) frame no.167 (C) frame no.200 (D) frame no.233 (E) frame no.266 (F) frame no.299 (G) frame no.432

Table 4.1: Quantitative results for some test sequences (Frames Tracked)

	3D Bha.	1D Bha.	1D Diffusion	1D EMD
Through red light	395/395	242/395	395/395	395/395
Through white light	58/92	13/92	92/92	66/92
Shadow	440/440	343/440	440/440	440/440
Flicking light	145/145	78/145	145/145	145/145
Dark hall	313/334	334/334	334/334	334/334
Average	96%	72%	100%	98%

Table 4.2: Quantitative results for some test sequences (Position Error)

	3D Bha.	1D Bha.	1D Diffusion	1D EMD
Through red light	16.876	124.248	**11.305**	17.685
Through white light	36.164	107.538	**4.382**	18.371
Shadow	19.722	17.005	10.057	**9.467**
Flicking light	5.207	11.881	**4.909**	5.809
Dark hall	23.117	18.439	**7.796**	8.806

stops at the black window because the color of this region is similar to the target object model (white and black). However this kind of problem can be solved by choose $\alpha = 0.4$ here. Whatever we choose $\alpha = 0.4$ or $\alpha = 0$, the result is good when the object go through the gray shadow from (H) to (J).

4.2 Hardware Implementation Result and Analysis

4.2.1 Design Flow

The chip implementation for the proposed SLP-SISP follows the standard cell based design flow as shown in Fig. 4.4. In the following sections, a brief introduction to each part will be mentioned with the example of our design if necessary.

Algorithm Analysis

All the hardware design should start from the algorithm analysis. The algorithm analysis includes run time profiling, hardware mapping consideration, algorithm innovation and spec estimation, etc. A ”Hardware C” code is designed to simulate the hardware behavior in a certain level with ”C” programming language. This Hardware C can be used to analysis the result when we just use fix-point and some data truncated by the limited data storage. Also, Hardware C can be used to estimate the amount of cycles the hardware needs to use to complete the computation.

Architecture Design

After the hardware-like C model is verified to function correctly, the hardware architecture design begins. We apply Verilog Hardware Description Language (Verilog-HDL) to construct the system with module-lized design approach. After the programming in Verilog, we adopt Cadence NC-Verilog [47], and SpringSoft Verdi [48] as our simulation CAD tools. We can compare the result of hardware

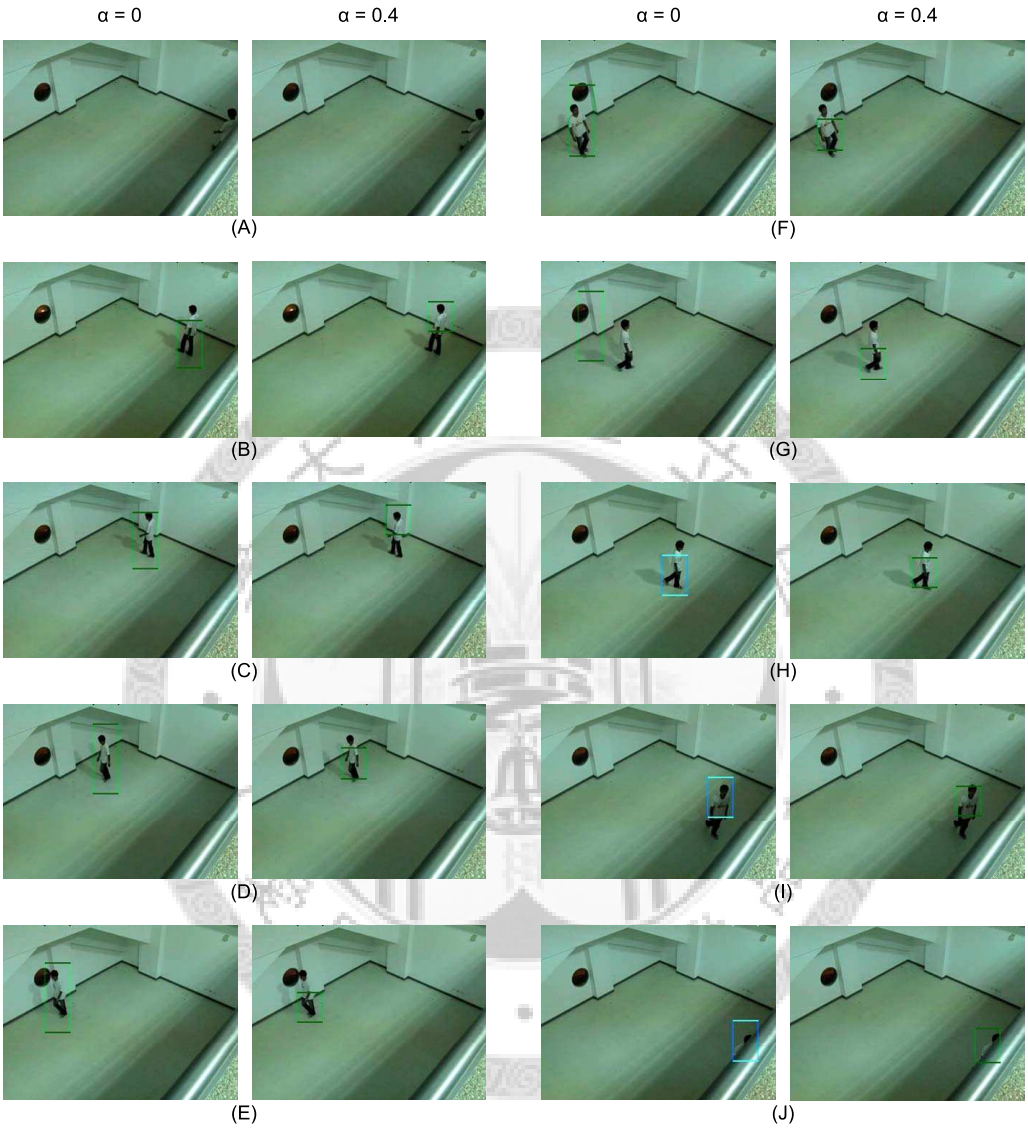


Figure 4.3: Compare $\alpha = 0$ (left) with $\alpha = 0.4$ (right). (A) frame no.40 (B) frame no. 100 (C) frame no. 160 (D) frame no.220 (E) frame no. 280 (F) frame no. 339 (G) frame no. 399 (H) frame no. 460 (I) frame no. 520 (J) frame no. 559.

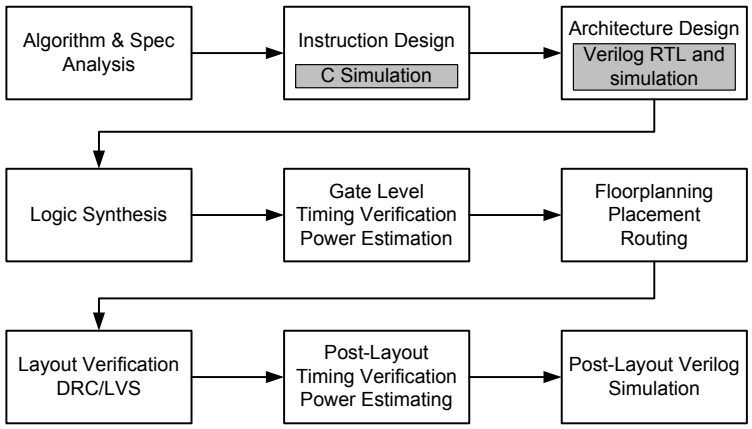


Figure 4.4: Cell-Based Design Flow

with the result from Hardware-C model.

Logic Synthesis

After checking the result of Verilog Simulation, we synthesis the design using Synopsys Design Compiler [49] with the UMC 90 nm cell library. According to our specifications for the chip, the area and timing are optimized in this stage. After logic synthesis, gate-level simulation is performed, and It can be done by using the NC-Verilog and Verdi with timing information of standard cells.

Gate Level Timing Verification and Power Estimation

Because the static timing analysis (STA) by Synopsys Design Compiler is not very precise, we use Synopsys Prime Time for more precise gate-level timing verification. The power is estimated with the Synopsys Prime Power, because the power estimated by Synopsys Design Compiler is not precise especially the gated-clock used. The switching power, the internal power, and the leakage power are analyzed with dynamic pattern input. By using the estimated power consumption, we could perform the power-plan of the chip to ensure that the power supply is enough and the power density is equally distributed on the chip.

Floorplanning, Placement and Routing

Cadence SoC Encounter is used as the tool of floorplanning, automatic placement and routing to generate the chip layout from gate-level netlists. In order to guarantee the timing specification is met, we use timing driven place-and-route of SoC Encounter.

Layout Verification

After the chip layout is obtained, it should pass the Design Rule Checking (DRC) and Layout Versus Schematic (LVS) by Calibre [50]. Drc is used to verify if the layout follows the manufacture rules, whereas LVS is used to compare the layout with the given netlists.

Post Layout Simulation, Timing Verification, and Power Estimation

We use NC-Verilog and Verdi to perform post-layout gate-level simulation with the extracted parasitic resistance and capacitance parameters to confirm the timing after clock tree synthesis, gate sizing and buffer insertion done by SoC Encounter. Moreover, some logic gates are inserted in the timing-driven P&R, and the actual clock tree is generated after the P&R, too. Therefore, the timing and the power should be verified and estimated again by prime Time and Prime Power, respectively.

4.2.2 Chip Layout and Specification

The design goal of our chip implementation is listed as follows: 640×480 screen size and 4:2:0 YUV color. The chip is designed in cell-based design flow with UMC 90nm 1P9M standard cell library. The chip layout is shown in 4.5, The specification is shown in 4.3.

The core size is $2.167 \times 2.167 mm^2$. A 64-bits bus width is assumed for the input and output pads. The designed work frequency is 125MHz

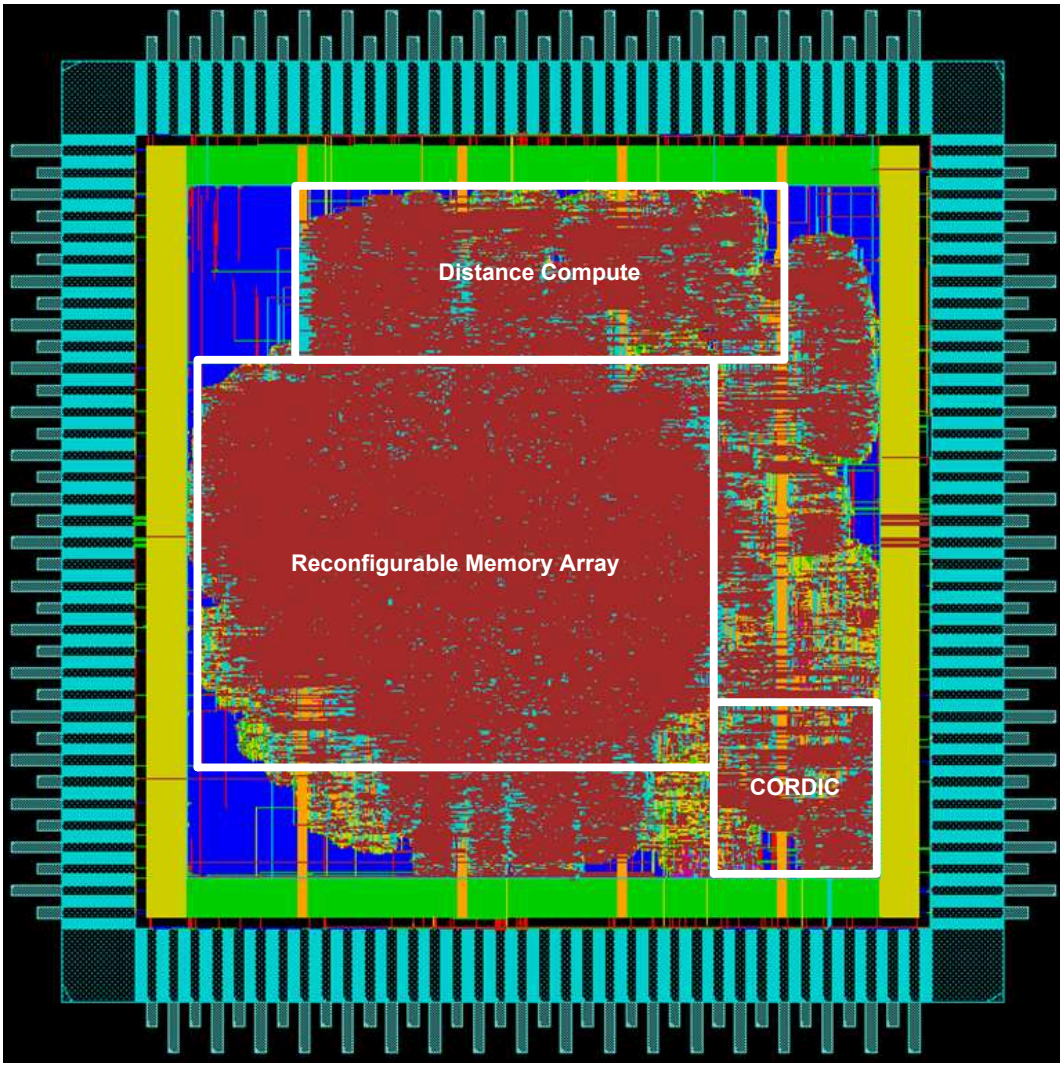


Figure 4.5: Chip layout of SLP-SISP

Table 4.3: Chip specification

Circuit name	SLP-SISP
Technology	UMC 90nm 1P9M
Package	144CQFP
Chip size	$3.260mm \times 3.260mm$
Core size	$2.167mm \times 2.167mm$
Gate count	680,716(2-1 NAND)
Working frequency	125MHz
Power consumption	33.3mW
Input pad	34
Output pad	1
InOut pad	64
Power pad	45

4.2.3 Summary and Comparison

The number shown in Table 4.5 is the maximum number of objects we can track when we use 640×480, 4:2:0 YUV color, 30 fps test sequence. The first column is the method we accumulation the histograms, with or without background color removal. The second column is the particle size be used or the object size. The third column is the maximum number of objects can be tracked with the number of particle on each object shown in the fourth column . The max size of the particle is about 80×80. However, if we want to track a larger object, we can perform down-sampling on the input sequence. The maximum number of objects when background color removal is not using is 11 and the maximum bandwidth of the CPU we use is 256MHz.

With SLP, 1-bit data stream, 8-bit data stream, 16-bit data stream, and 32-bit data stream can share the same memory, which is our Reconfigurable Memory Array. From Table 4.4, without SLP, the memory usage is about 77344 bits, and

with SLP, the memory usage is about 30720 bits. The overall saving is about 60.28%. This shows that the SLP is very useful in hardware sharing, especially for on chip memory.

First, we compare our design with a product developed by ObjectVideo [51]. This company is a provider of intelligent video software for security safety, business intelligence, process improvement and other applications The ObjectVideo onboard is used with a DSP or CPU, and it needs 20% to 25% utilization of TI DM642DSP at 720 MHz frequency or 12% of a single core on an Intel Core 2 Duo T7400 at 2.16GHz frequency. However we do not have the spec of its hardware, we can only compare our spec with the DSP it uses. It can perform video analysis on 10 fps of frame resolution 352×288, YUV 4:2:2 sequences.

Second, we show the performance of segmentation on SLP-SISP in Table 4.6. If we use our chip to do the segmentation algorithm only, we can handle higher frequency sequences. We also use PC to simulate the required time on the CPU, and the CPU here is Intel Core 2 Duo CPU E8400 at frequency 3GHz.

Third, we can compare the tracking hardware with others' work here. Table 4.7 is the comparison of tracking hardware. Kristensen et al. [52] use Xilinx Virtex II pro vp30 FPGA development to implement their design. However, in this implementation, they only use grey level image. Although they can track more objects at the same time, they need a higher frequency clock and lower frame resolution. In this table, we can know that our computation time is better.

Table 4.4: Memory saving

Without SLP Memory Sharing	77344 bits
With SLP Memory Sharing	30720 bits
Saving	60.28%

Table 4.5: The maximum number of objects can be tracked in 30 fps sequences

Histogram Accumulation	Particle size	Objects	Particles
With background color removal	50×50	10	100
With background color removal	80×80	7	100
Without background color removal	50×50	11	100
Without background color removal	80×80	11	100

Table 4.6: The comparison of segmentation hardware

	Our	CPU
Frame resolution	640×480	640×480
Frame frequency	125 fps	7.04 fps
Working frequency	125MHz	3GHz
Color	YUV 4:2:0	YUV 4:2:0

Table 4.7: The comparison of tracking hardware

	Our	Kristensen et al.	CPU
Frame resolution	640×480	320×240	640×480
Frame frequency	30 fps	25 fps	3.2 fps
Number of objects	10	61	1
Working frequency	125MHz	1GHz	3GHz

Chapter 5

Conclusion

We design a robust tracking algorithm that can handle two common problems in tracking video object: object appearance changes due to lighting condition changes and tracking video objects through similar-colored background. We have also developed a hardware coprocessor SLP-SISP that can efficiently accelerate most of the operations in intelligent surveillance algorithms.

In video object tracking, we proposed an particle filter framework. In this framework, the particle filter is enhanced with diffusion distance for histogram comparison and the segmentation result is also employed to determine the particle weight. In other words, we use both the color histogram and segmentation results to determine the final locations of target objects. In our experiments, our method with diffusion distance outperforms the typical particle filter with Battacharyya distance under lighting condition changes. Moreover, the segmentation results is also proved to be helpful in tracking video object through similar-colored background.

In the proposed hardware coprocessor SLP-SISP, streaming based processing, sub-word level parallelism and hardware sharing techniques are employed together to achieve high throughput, low memory bandwidth, programmability, and low cost. In our results, the on-chip memory saving is up to 60.28% with SLP. The hardware coprocessor can be used to accelerate intelligent surveillance

functions, such as segmentation, tracking, motion analysis, feature detection and feature description. In our implementation with UMC 90nm technology, we can do video object segmentation and, at the same time, track 10 objects in frame size 640×480, YUV 4:2:0, 30 fps sequence with 125 MHz clock frequency and 33.3mW power consumption.

In the future, we plan to support more vision algorithms with our coprocessor.

Reference

[1] D. Cormaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 26, no. 11, pp. 1531–1536, Nov. 2004.

[2] Z. Yin and R. Collins, "Spatial divide and conquer with motion cues for tracking through clutter," in *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, June 2006, vol. 1, pp. 570–577.

[3] A. Yilmaz and X. Li, "Contour-based object tracking with occlusion handling in video acquired using mobile cameras," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 26, no. 11, pp. 1531–1536, Nov. 2004.

[4] D. Chen and J. Yang, "Robust object tracking via online dynamic spatial bias appearance models," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 29, no. 12, pp. 2157–2169, Dec. 2007.

[5] Y.-T. Chen, C.-S. Chen, and Y.-P. Hung, "Integration of Background Modeling and Object Tracking," in *Proc. IEEE Int. Conf. Multimedia and Expo*, July 2006, pp. 757–760.

[6] Z. Wang, X. Yang, Y. Xu, and S. Yu, "Camshift guided particle filter for visual tracking," in *Proc. IEEE Signal Processing Systems. Workshop*, Oct. 2007, pp. 301–306.

[7] F. Liu, Q. Liu, and H. Lu, "Robust color-based tracking," in *Proc. IEEE Signal Processing Systems. Workshop*, Oct. 2007, pp. 301–306.

[8] M. Jaward, L. Mihaylova, N. Canagarajah, and D. Bull, "Multiple object tracking using particle filters," in *Proc. IEEE Conf. Aerospace*, 2006.

[9] E. Maggio, F. Smerladi, and A. Cavallaro, "Adaptive multifeature tracking in a particle filtering framework," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 10, pp. 1348–1359, Oct. 2007.

[10] S. K. Zhou and R. Chellappa, "Visual tracking and recognition using appearance-adaptive models in particle filters," *IEEE Trans. Image Processing*, vol. 13, no. 11, Nov. 2004.

[11] P. Pan and D. Schonfeld, "Dynamic proposal variance and optimal particle allocation in particle filtering for video tracking," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, pp. 1268–1279, Sept. 2008.

[12] H. Wang, D. Suter, K. Schindler, and C. Shen, "Adptive object tracking based on an effective appearance filter," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 29, no. 9, pp. 1661–1667, Sept. 2007.

[13] H. Jiang, S. Fels, and J. J. Little, "Optimizing multiple object tracking and best view video synthesis," *IEEE Trans. Multimedia*, vol. 10, pp. 997–1012, Oct. 2008.

[14] H. Jiang, S. Fels, and J. J. Little, "A linear programming approach for multiple object tracking," in *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[15] T.-H. Tsai, Y.-C. Liu, D.-M. Chen, and C.-Y. Lin, "Fast occluded object tracking technique with distance evaluation," in *Proc.* 9^{th} *Joint Conf. Information Sciences*, Oct. 2006.

[16] W. Hu, X. Zhou, M. Hu, and S. Maybank, "Occlusion reasoning for tracking multiple people," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 1, pp. 114–121, Jan. 2009.

[17] W.-K. Chan and S.-Y. Chien, "Real-time memory-efficient video object segmentation in dynamic background with multi-background registeration technique," in *Proc. IEEE Multimedia Signal Process. Workshop*, July 2007.

[18] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, July 1999, pp. 246–252.

[19] D. G. Lowe, "Distinctive image features form scale-invariant keypoints," *Int. J. of Computer Vision*, 2004.

[20] T.-W. Chen, W.-K. Chan, and S.-Y. Chien, "Efficient face detection with segmentation and feature-based face scoring in surveillance systems," in *Proc. IEEE Multimedia Signal Processing Workshop*, Oct. 2007, pp. 215–218.

[21] A. A. Abbo, R. P. Kleihorst, V. Choudhary, L. Sevat, P. Wielage, S. Mouy, B. Vermeulen, and M. Heijligers, "Xetal-II: A 107 GOPS, 600 mW massively parallel processor for video scene analysis," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 192–201, Jan. 2008.

[22] H. Noda et al, "The design and implementation of the massively parallel processor based on the matrix architecture," *IEEE J. Solid-State Circuits*, vol. 42, no. 1, pp. 804–812, Apr. 2007.

[23] M. Nakajima et al, "A 40 GOPS 250 mW massively parallel processor based on matrix architecture," in *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*, 2006, pp. 410–411.

[24] K. Mizumoto et al, "A multi matrix-processor core architecture for real-time image processing SoC," in *Proc. IEEE Asian Solid-State Circuit Conf.*, Nov. 2007.

[25] S. Kaneko et al, "A 600-MHz single-chip multiprocessor with 4.8-GB/s internal shared pipelined bus and 512-kB internal memory," *IEEE J. Solid-State Circuits*, vol. 39, no. 1, pp. 184–193, Jan. 2004.

[26] S. Kyo, T. Koga, and S. Okazaki, "IMAP-CE : A 51.2 GOPS video rate image processor with 128 VLIW processing elements," in *Proc. IEEE Int. Conf. Image Processing*, Sept. 2001, vol. 3, pp. 294–297.

[27] S. Kyo, T. Koga, and S. Okazaki, "A 51.2-GOPS scalable video recognition processor for intelligent cruise control based on a linear array of 128 four-way VLIW processing elements," *IEEE J. Solid-State Circuits*, vol. 38, no. 11, pp. 1992–2000, Nov. 2003.

[28] S. Kyo, S. Okazaki, and T. Arai, "An integrated memory array processor for embedded image recognition systems," *IEEE Trans. Comput.*, vol. 56, no. 5, pp. 622–634, May 2007.

[29] B. K. Khailany, T. Williams, J. Lin, E. P. Long, M. Rygh, D. W. Tovey, and W. J. Dally, "A programmable 512 GOPS stream processor for signal, image, and video processing," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 202–213, Jan. 2008.

[30] B. Serebrin, J. D. Owens, C. H. Chen, S. P. Crago, U. J. Kapasi, B. Khailany, P. Mattson, J. Namkoong, S. Rixner, and W. J. Dally, "A stream processor development platform," in *Proc. IEEE Int. Conf. Computer Design: VLSI in Computers and Processors*, Sept. 2005, pp. 303–308.

[31] U. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany, "The imagine stream processor," in *Proc. IEEE Int. Conf. Computer Design*, Sept. 2002, pp. 282–288.

[32] S. Ciricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, and A. Saidi, "The reconfigurable streaming vector processor ($RSVP^{TM}$)," in *Proc. the* 36^{th} *Int. Symp. Microarchitecture*, 2003.

[33] S. Chiricescu, M. Schuette, R. Essick, B. Lucas, P. May, K. Moat, and J. Norris, "*RSVP*^{TM}: An automotive vector processor," in *Proc. Intelligent Vehicles Symp.*, 2004.

[34] S. M. Chai, S. Chiricescu, R. Essick, B. Lucas, P. May, K. Moat, J. M. Norris, and M. Schuette, "Streaming processors for next-generationi mobile imaging application," *IEEE Communication Magazine*, pp. 81–89, Dec. 2005.

[35] S. Chiricescu, S. M. Chai, K. Moat, B. Lucas, P. May, J. Norris, R. Essick, and M. Schuette, "RSVP II: A next generation automotive vector processor," in *Proc. Intelligent Vehicles Symp.*, 2005, pp. 563–568.

[36] M. B. Taylor et al, "Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ILP and streams," in *Proc.* 31st *Annual International Symp. Computer Architecture*, June 2004, pp. 2–13.

[37] *http://en.wikipedia.org/wiki/Particle_filter*.

[38] M. Muhlich, "Particle filters an overview," in *Proc. Filter-Workshop*, 2003.

[39] A. Bhattacharyya, "On a measure of divergency between two statistical populations defined by their probability distributions," *Bulletin of the Calcutta Mathematical Society 35*, pp. 99–109, 1943.

[40] *http://en.wikipedia.org/wiki/Bhattacharyya_distance*.

[41] Y. Rubner, L. Guibas, and C. Tomasi, "The earth mover's distance, multi-dimension scaling, and color-based image retrieval," in *Proc. ARPA Image Understanding. Workshop*, May 1997.

[42] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metrix for image retrieval," *Int. J. Computer Vision*, pp. 99–121, 2000.

[43] H. Ling and K. Okada, "Diffusion distance for histogram comparison," in *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2006.

[44] H. Ling and K. Okada, "An efficient earth mover's distance algorithm for robust histogram comparison," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 29, no. 5, pp. 840–863, May 2007.

[45] *http://www.arm.com*.

[46] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proc. ACM/SIGDA Sixth Int. Symp. Field Programmable Gate Arrays (FPGA-98)*, Feb. 1998, pp. 192–200.

[47] *http://www.cadence.com*.

[48] *http://www.springsoft.com*.

[49] *http://www.synopsys.com*.

[50] *http://www.mentor.com*.

[51] *http://www.objectvideo.com*.

[52] F. Kristensen, H. Hedberg, H.u Jiang, P. Nilsson, and V. Öwall, "An embedded real-time surveillance system: Implementation and evaluation," *J. Signal Processing Systems*, vol. 52, no. 1, pp. 75–94, July 2008.