Efficient Arithmetic of Gaussian Mixture Model for Pattern Recognition and Machine Learning

A Dissertation
in
Electrical and Information Engineering
Submitted to Graduate School of Science and Technology
Niigata University
in
Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

Hidenori Watanabe

March 2013

Acknowledgment

I would like to thank advisory professors, Dr. Shogo Muramatsu, Dr. Shigenobu Sasaki, Dr. Hisakazu Kikuchi and Dr. Masahiro Yukawa for their guidance, patience and support throughout my undergraduate and graduate research activities with every aspect. I would like to thank the committee member of this thesis, Prof. Keisuke Nakano. His valuable comments was very helpful for the progress in this thesis.

I would like to acknowledge the support from the Strategic Information and Communications R&D Promotion Programme (SCOPE), No. 102304003, and from a contract of research and development for radio resource enhancement, Ministry of Internal Affairs and Communications, Japan.

I would additionally like to thank Dr. Minoru Hiki, Mr. Tsutomu Watanabe, Mr. Ryunosuke Takeda, Mr. Takahiro Sato, Mr. Koji Nagumo, Mr. Yuji Kikuchi, and Mr. Mitsuru Takahashi, for encouragements and supports. This thesis would not have been possible without their supports. Although there have appeared no names here, I thank to all individuals in Department of Electrical and Electronic Engineering and Electronic Engineering and Graduate School of Science and Technology, Niigata University, my friends and relatives for their generous support. Finally, I would like to my special appreciation to my family.

March 2013
Hidenori Watanabe

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Aim of This Thesis . . . 2
- 1.3 Organization . . . 2

2 Interval Calculation for GMM **5**
- 2.1 Review of GMM-based Classification . . . 5
  - 2.1.1 Bayesian decision rule . . . 5
  - 2.1.2 Gaussian mixture model(GMM) . . . 7
  - 2.1.3 GMM-based classification by Bayesian decision rule . . . 7
- 2.2 Interval Calculation for GMM-Based Classification . . . 8
  - 2.2.1 Initial decision process . . . 8
  - 2.2.2 Refinement process . . . 10
  - 2.2.3 Termination process . . . 12
  - 2.2.4 Proposed algorithm . . . 14
- 2.3 Performance Evaluation . . . 15
  - 2.3.1 Experiments with color extraction . . . 15
  - 2.3.2 Computational cost on DSP . . . 17
  - 2.3.3 Computational cost on FPGA . . . 18
- 2.4 Summary . . . 21

3 Narrowed Initial Interval **23**
- 3.1 Narrowed Initial Interval . . . 23
  - 3.1.1 Calculation of shift amount . . . 23
  - 3.1.2 Look-up table for multiplierless interval calculation . . . 25
  - 3.1.3 Refinement process with bit shift and look-up table . . . 25
  - 3.1.4 Decision rule in the proposed refinement scheme . . . 26
- 3.2 Performance Evaluation on FPGA . . . 27
  - 3.2.1 Implementation condition . . . 27
  - 3.2.2 Synthesis report . . . 27
- 3.3 Summary . . . 28

4 LUT-Based GMM **31**
- 4.1 Review of EM Algorithm for GMM . . . 31
  - 4.1.1 E-step . . . 31

4.1.2 Computation of EM algorithm for GMM 32
4.2 EM Algorithm with Look-Up-Table-Based Exponential Function . . 33
4.2.1 Look-Up-Table-Based Exponential Function 33
4.2.2 Scale adjustment for look-up table 34
4.2.3 Scaling using the weighted average of LUT entries 34
4.3 Performance Evaluation 35
4.3.1 Precision of parameter estimation 36
4.3.2 Computational speed 37
4.4 Summary 38

5 Conclusions **41**
5.1 Summary 41
5.2 Open Problems 42

Appendix A: Derivation of Eq. (2.15) **43**

List of Figures **45**

List of Tables **47**

Bibliography **51**

Biography **53**

Chapter 1

Introduction

1.1 Background

A Gaussian mixture model (GMM) is a probability density function for continuous variable. The GMM is expressed in a weighted sum of Gaussian distribution and often used in which case assumed distribution is complicated form such as multimodal distribution. Since the GMM has the flexibility originate in the form of weighted sum, the GMM is widely used in various study areas. In image processing, the GMM is often used as background or foreground model for image segmentation, such as skin detection and background subtraction, [1, 2, 3, 4, 5]. The applications of the GMM for speech processing are gender classification [6], speaker verification [7, 8], speech recognition [9], and so on. The classification of gas is also an examples of GMM-based application[10]. As described above, the GMM is an useful probability density function for pattern recognition and machine learning.

A problem in pattern recognition and machine learning techniques is the computational cost. Although the image segmentation for monitoring camera and speech recognition on smart phone are useful applications of pattern recognition, the implementation of these pattern recognition is difficult when target device is embedded systems. Especially, the resources, such as battery and processor, are constrained on embedded systems. For long-life and real-time operation, the computational cost of pattern recognition should be reduced. Machine learning on embedded system is also studied. Usually, training data are collected for a server and parameters are estimated on high-powered computers. However, Gu proposed a distributed machine learning technique for sensor network[11]. The proposed method estimates the parameters of GMM on each sensor nodes since communication resources on sensor network is limited[12]. As described above, the computational complexity of the GMM should be reduced to diffuse the use of beneficial GMM applications in pattern recognition and machine learning.

1.2 Aim of This Thesis

The aim of this thesis is to propose efficient arithmetic of the GMM focuses on exponential function.

There are some viewpoints to reduce the computational cost of the GMM. For example, fixed-point implementation is a well-known method for reducing the computational cost. For GMM, moreover, the distributed arithmetic for quadratic form was proposed by Shi et al. [10].

For further reduction of the computational cost of the GMM, this thesis focuses on exponential function. This thesis shows two approaches to approximate exponential function. One is interval calculation. The interval calculation is generally used to guarantee the accuracy of computation. However, this study satisfies both of the computational cost and accuracy. Second is the look-up table (LUT) based implementation. The exponential function computed by LUT is not uncommon method, but this study provides flexibility for the values in LUT under certain conditions.

In this thesis, two applications of the GMM are used to confirm the significance of these approaches. One is the GMM classification based on Bayesian decision rule. The interval calculation method is evaluated on the GMM classification. The other is the EM algorithm for GMM parameter estimation. The LUT based method is evaluated on the EM algorithm. These applications are evaluated on digital signal processor (DSP), field programmable gate array (FPGA) or PC.

1.3 Organization

This section shows the organization of this thesis. Figure 1.1 illustrates the relation between chapters. This thesis is organized as follows:

Chapter 2: Interval calculation for GMM

Chapter 2 proposes an efficient arithmetic with an interval representation of the GMM. This chapter indicates the possibility that the computation of the GMM does not always require high-accuracy calculation for any inputs. Then, a classification method with interval calculation is proposed. The three processes, called initial decision, refinement and termination process, in interval calculation method are introduced in this chapter. From some simulation, it is shown that the interval calculation method is able to classify inputs efficiently.

The GMM classification based on Bayesian decision rule is also reviewed in this chapter.

Chapter 3: Narrowed initial interval

Chapter 3 proposes an advanced interval representation. The initial decision process in Chapter 2 calculates coarse interval. If the range of initial interval is narrow, the

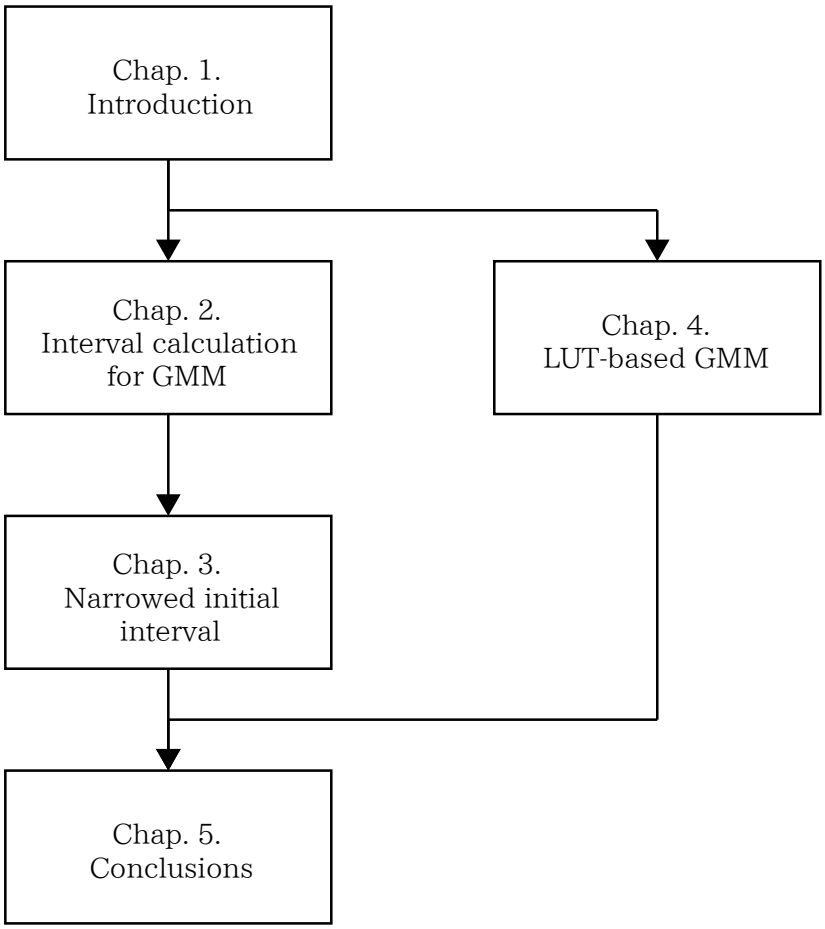


Figure 1.1: Organization of this thesis. Chapter 2 and 3 focus on balancing the computational cost with the computational accuracy. Chapter 3 improves the refinement process in Chapter 2. Chapter 4 focuses on flexibility for LUT. Chapter 5 presents conclusions.

number of iteration can be reduced. This chapter shows how to improve the range of initial interval and simulation results.

Chapter 4: LUT-based GMM

Chapter 4 proposes a LUT-based approximation for exponential function in GMM. This work provides flexibility for the values in LUT under certain conditions. Through the use of the flexibility, it is shown that the bit length of LUT can be reduced. The EM algorithm for parameter estimation of the GMM is used for evaluation of the LUT-based implementation of the GMM.

The EM algorithm is also reviewed in this chapter.

Chapter 5: Conclusions

In Chapter 5, some conclusions are drawn after a summary of this study is given.

Chapter 2

Interval Calculation for GMM

This chapter proposes interval representation for GMM. Moreover, for evaluation, the interval is applied for GMM-based classification.

The approximation methods for exponential function, generally, always calculates high-precision values for any inputs. However, high-precision computation is not required in some GMM applications. Figure 2.1 illustrates a concept of GMM-based classification. In the neighborhoods of the point of intersection between tow lines, such as squared area, high-precision computation is required since the values are compared. However, other areas clearly far away from the other line. Therefore, the computational cost of the GMM can be reduced in such regions.

Following sections proposes GMM-based classification with interval calculation as an application of the interval approach.

2.1 Review of GMM-based Classification

This section reviews the Bayesian decision rule, GMM, and GMM-based classification.

2.1.1 Bayesian decision rule

Bayesian decision rule enables us to identify the class of an observed datum or feature vector $\mathbf{x}$ provided possible classes are known a priori. Given a feature vector $\mathbf{x}$, Bayesian decision scheme compares the posterior probabilities of every class, i.e. $P[c = C_k|\mathbf{x}]$, and then select the class of the highest probability. This decision is known to give the smallest error rate [13, 14]. Since the Bayesian theorem tells us $P[c|\mathbf{x}] \propto p(\mathbf{x}|c)P[c]$, the Bayesian decision can be reduced to the evaluation with the following discriminant function:

$$f_{k,l}(\mathbf{x}) = p(\mathbf{x}|C_k)P[C_k] - p(\mathbf{x}|C_l)P[C_l], \tag{2.1}$$

where $p(\mathbf{x}|c)$ is the marginal probability and P[c] is the priori probability. If $f_{k,l}(x) > 0$, then Class C_l is dropped from the candidates. If $f_{k,l}(\mathbf{x}) < 0$, then Class C_k remains nominated.

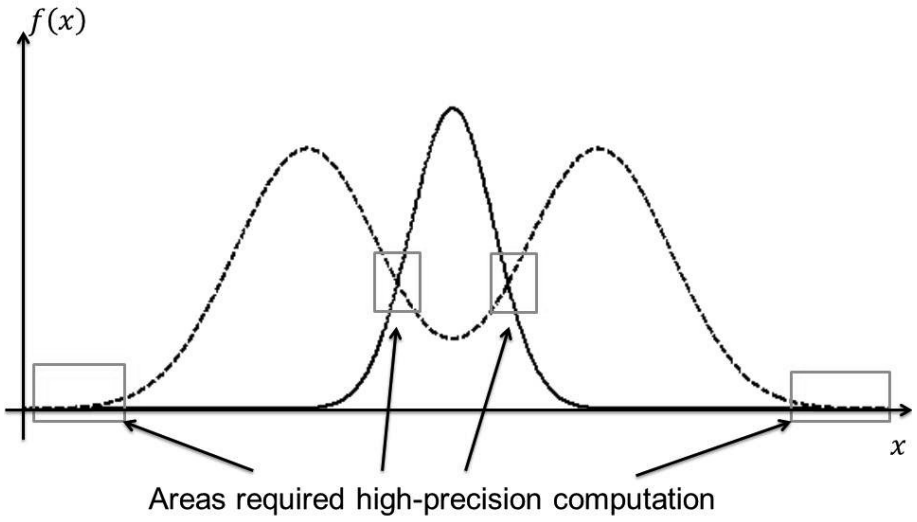


Figure 2.1: An example of GMM-based classification. The dotted line shows a GMM and the solid line shows a Gaussan distribution. The inputs are classified as the class that the value of functions is large. In the neighborhoods of the point of intersection between tow lines, such as squared area, are required high-precision computation. However, other areas clearly far away from the other line.

2.1.2 Gaussian mixture model(GMM)

Equation (2.1) requires us to know the conditional density $p(\mathbf{x}|c)$ and P[c] a priori. This work assumes GMM for $p(\mathbf{x}|c)$ [14]. Let $\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma})$ be the density of Gaussian distribution. $\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma})$ is defined by

$$\mathcal{N}(\mathbf{x}|\mu,\mathbf{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\mathbf{\Sigma}|^{\frac{1}{2}}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right\}, \tag{2.2}$$

where D is the number of variables, μ and $\mathbf{\Sigma}$ are a $D \times 1$ mean vector and $D \times D$ covariance matrix, respectively. Then, the density of GMM, $\mathcal{M}(\mathbf{x}|\mathbf{\Theta})$, is represented by

$$\mathcal{M}(\mathbf{x}|\mathbf{\Theta}) = \sum_{n=0}^{N-1} \alpha_n \mathcal{N}(\mathbf{x}|\mu_n, \mathbf{\Sigma}_n), \tag{2.3}$$

where Θ is a set of parameters, that is $\Theta = \{\{\alpha_n\}, \{\mu_n\}, \{\mathbf{\Sigma}_n\}\}$, N is the number of Gaussian distributions and α_n is the mixture ratio of the n-th distribution. The mixture ratios satisfy the condition $\sum_{n=0}^{N-1} \alpha_n = 1$ and $0 \leq \alpha_n \leq 1$.

2.1.3 GMM-based classification by Bayesian decision rule

Since the probability of $\mathbf{x}$ given c, i.e. $p(\mathbf{x}|c)$, is assumed to be GMM in Eq. (2.3), Eq. (2.1) is represented by

$$f_{k,l}(\mathbf{x}) = g_k(\mathbf{x}) - g_l(\mathbf{x}), \tag{2.4}$$

$$g_k(\mathbf{x}) = \sum_{n=0}^{N_k-1} K_{k,n} \exp(-z_{k,n}), \tag{2.5}$$

where

$$z_{k,n}(\mathbf{x}) = \frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu), \tag{2.6}$$

$$K_{k,n} = \frac{P[C_k]\alpha_{k,n}}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}, \tag{2.7}$$

where $\{\alpha_{k,n}\}$, $\{\mu_{k,n}\}$ and $\{\mathbf{\Sigma}_{k,n}\}$ are parameter sets of GMM for Class C_k [10]. Since the covariance matrix $\mathbf{\Sigma}_{k,n}$ is positive definite, function $z_{k,n}(\mathbf{x})$ obtained by the quadratic form in Eq. (2.6) is guaranteed to be non-negative. As well, the conditions on $\alpha_{k,n}$ guarantee for constant $K_{k,n}$ to be non-negative. Therefore, Eq. (2.6) must be non-negative and $g_k(\mathbf{x})$ ranges from 0 to $\sum_{n=0}^{N_k-1} K_{k,n}$. For the sake of convenience, function $z_{k,n}(\mathbf{x})$ is simply represented by variable $z_{k,n}$ in the following discussion.

2.2 Interval Calculation for GMM-Based Classification

On the assumption that a feature vector $\mathbf{x}$ is drawn from a Gaussian distribution, the Bayesian decision is achieved by comparing scaled exponential functions, i.e. $K\exp(-z)$, where K and z are nonnegative constant and variable. Usually, the comparison is reduced by taking their logarithms, i.e. $\ln K\exp(-z) = \ln K - z$, where $\ln K$ is a constant. Note that the classifier does not require any exponential function. On the other hand, it is not true for GMM. In this section, we propose a simple interval calculations and an adaptive control of computational precision for the exponential operations in GMM. The following facts are used in our proposal.

- Required result is only the sign of Eq. (2.4).
- Constant $K_{k,n}$ and variable $z_{k,n}$ in Eq. (2.5) are all non-negative.

2.2.1 Initial decision process

First of all, we give an inequality which represents an interval covering the function in Eq. (2.5) with integer powers of two. From the fact that $\exp(-z) = 2^{-z\log_2 e}$ and $z \geq 0$, we have the relation

$$2^{-(\lfloor z\log_2 e\rfloor+1)} < \exp(-z) \leq 2^{-\lfloor z\log_2 e\rfloor}, \tag{2.8}$$

where $\lfloor x\rfloor$ is the integer part of x. The relation expressed in Eq. (2.8) is shown in Fig. 2.2.

Furthermore, Eq. (2.8) leads

$$\begin{aligned}\sum_{n=0}^{N-1} K_n 2^{-(\lfloor z_n\log_2 e\rfloor+1)} &< \sum_{n=0}^{N-1} K_n\exp(-z_n)\\ &\leq \sum_{n=0}^{N-1} K_n 2^{-\lfloor z_n\log_2 e\rfloor}.\end{aligned} \tag{2.9}$$

The lower and upper bound of this interval is calculated only by scaling constant K_n with an integer power of two and accumulating the results. If K_n is in a fixed-point representation, this operation is executed by the right bit-shift. If it is in a floating-point one, the index decrement realizes the computation. Figure 2.3 illustrates a case that the class C_0 is clearly delated from nominations with the interval computation, where we define

$$g_k^{\text{upper}}(\mathbf{x}) = \sum_{n=0}^{N_k-1} K_{k,n} 2^{-\lfloor z_{k,n}\log_2 e\rfloor}, \tag{2.10}$$

$$g_k^{\text{lower}}(\mathbf{x}) = \sum_{n=0}^{N_k-1} K_{k,n} 2^{-(\lfloor z_{k,n}\log_2 e\rfloor+1)} = \frac{1}{2} g_k^{\text{upper}}(\mathbf{x}). \tag{2.11}$$

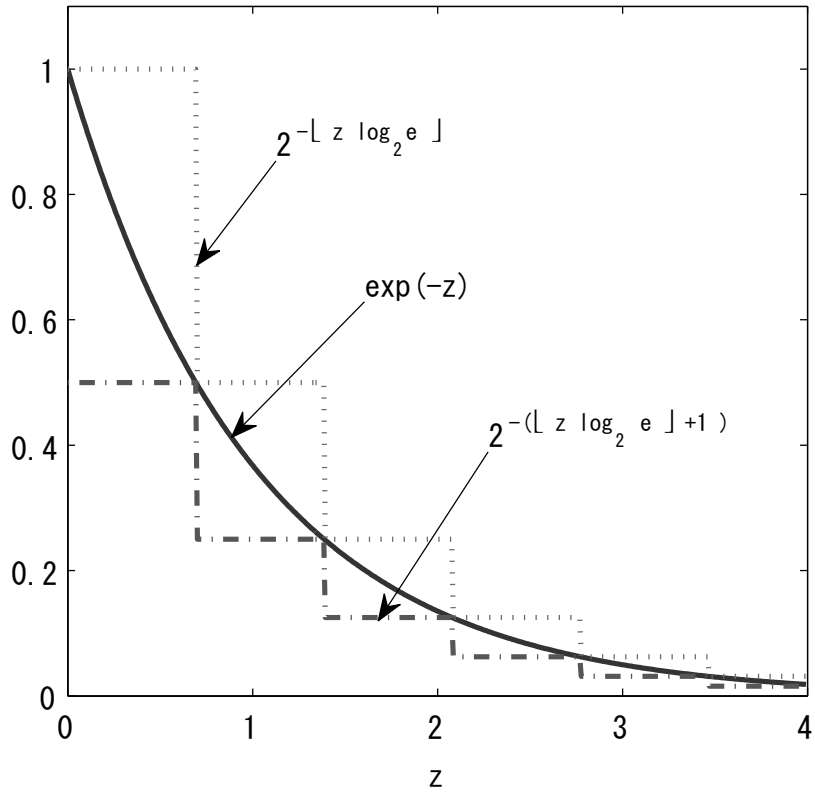


Figure 2.2: Relation of the inequality in Eq. (2.8).

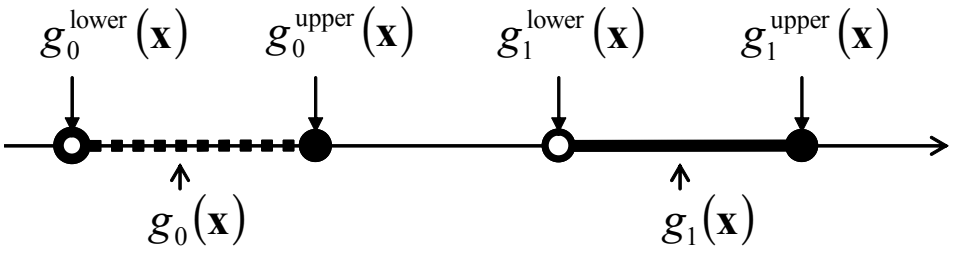


Figure 2.3: Case that the interval is used for classification.

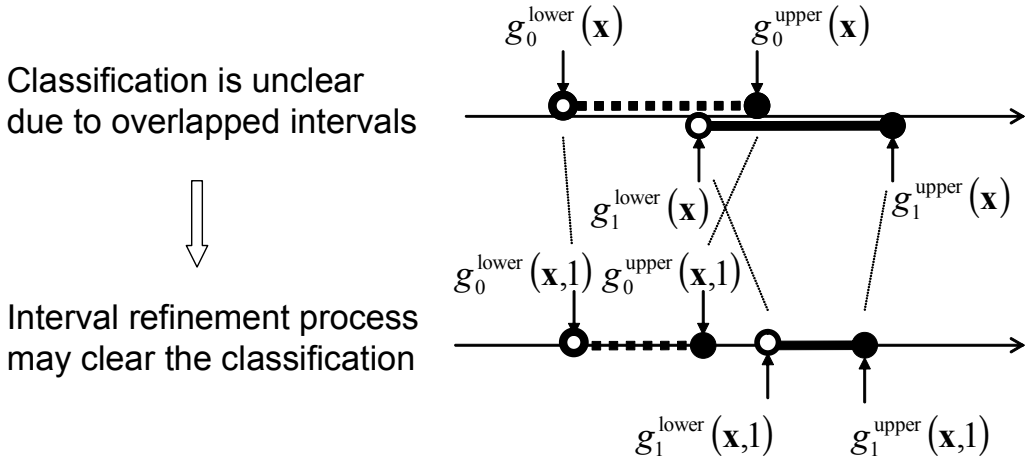


Figure 2.4: Case that the interval computation is insufficient (top). An expected effect of the refinement process (bottom).

Note that the lower bound in Eq.(2.11) is a half of the upper one and this fact is important in terms of computational cost. Each interval guarantees to include the true value of $g_k(\mathbf{x})$. The above decision process makes it possible to avoid precise calculations.

For the two class case, the number of operations required for one initial decision in the fixed-point implementation is summarized as follows:

- Multiplications of the constant $\log_2 e$ with the variable $z_{k,n}$: $N_0 + N_1$,
- Rounding of $z_{k,n} \log_2 e$: $N_0 + N_1$,
- Multiple bit-shift of the constant $K_{k,n}$: $N_0 + N_1$,
- Cumulative additions: $N_0 + N_1$,
- One bit-shift of $g_k^{\text{upper}}(\mathbf{x})$: 2,
- Comparisons: 2.

For the floating-point implementation, decrement of the exponential part replaces to the bit-shift operations. Note that the typical implementation with the exponential function requires $N_0 + N_1$ exponential operations, $N_0 + N_1$ constant multiplications $N_0 + N_1$ accumulative additions and one comparison.

2.2.2 Refinement process

When the probability of C_k given $\mathbf{x}$ close to another, the initial decision process confuses. In the followings, we propose a refinement process for successively improving the computational precision and narrowing the interval. Figure 2.4 illustrates

a case that the decision is unclear from the interval and the refinement process improves each interval which includes the true value $g_k(\mathbf{x})$. Let us begin with the relation

$$\exp(-z) = 2^{-\lfloor z\log_2 e\rfloor}2^{-\beta} = 2^{-(\lfloor z\log_2 e\rfloor+1)}2^{(1-\beta)}, \tag{2.12}$$

where β is the fractional part of $z\log_2 e$ and in the range $0 \le \beta < 1$. Representing the i-th fractional bit of β as $\beta[i] \in \{0, 1\}$, we have

$$\beta = \sum_{i=1}^{L} \beta^{[i]} 2^{-i}, \tag{2.13}$$

where L denotes the number of fractional bits. Furthermore, defining $T[i] = 2^{-2^{-i}}$, we obtain the following expressions:

$$2^{-\beta} = 2^{-\sum_{i=1}^{L}\beta^{[i]}2^{-i}} = \prod_{i=1}^{L} 2^{-\beta^{[i]}2^{-i}} = \prod_{i=1}^{L} T[i]^{\beta^{[i]}} \tag{2.14}$$

$$2^{(1-\beta)} = 2^{2^{-L}} \prod_{i=1}^{L} 2^{-\overline{\beta}^{[i]}2^{-i}} = T[L]^{-1} \prod_{i=1}^{L} T[i]^{\overline{\beta}^{[i]}} \tag{2.15}$$

where $\overline{\beta}^{[i]}$ is the bit inverse of $\beta^{[i]}$ (see Appendix A).

Then, let us consider an update process for refining the interval. According to Eqs. (2.12)-(2.15), the upper and lower bound in Eqs. (2.10) and (2.11) are improved by using the following bounds for $i \ge 1$:

$$g_k^{\text{upper}}(\mathbf{x}, i) = \sum_{n=0}^{N_k-1} h_{k,n}^{\text{upper}}(\mathbf{x}, i), \tag{2.16}$$

$$g_k^{\text{lower}}(\mathbf{x}, i) = \sum_{n=0}^{N_k-1} h_{k,n}^{\text{lower}}(\mathbf{x}, i), \tag{2.17}$$

where

$$h_{k,n}^{\text{upper}}(\mathbf{x}, i) = \begin{cases} K_{k,n}2^{-\lfloor z_{k,n}\log_2 e\rfloor}, & i = 0, \\ K_{k,n}2^{-\lfloor z_{k,n}\log_2 e\rfloor}\prod_{j=1}^{i} T[j]^{\beta_{k,n}^{[j]}}, & i \ge 1, \end{cases} \tag{2.18}$$

$$h_{k,n}^{\text{lower}}(\mathbf{x}, i) = \begin{cases} K_{k,n}2^{-(\lfloor z_{k,n}\log_2 e\rfloor+1)}, & i = 0, \\ K_{k,n}2^{-(\lfloor z_{k,n}\log_2 e\rfloor+1)}\prod_{j=1}^{i} T[j]^{-\overline{\beta}_{k,n}^{[j]}}, & i \ge 1, \end{cases} \tag{2.19}$$

where $\beta_{k,n}^{[i]}$ denotes the i-th fractional bit of $z_{k,n}\log_2 e$.

For $i \ge 1$, we have the update equations

$$h_{k,n}^{\text{upper}}(\mathbf{x}, i) = h_{k,n}^{\text{upper}}(\mathbf{x}, i-1)T[i]^{\beta_{k,n}^{[i]}}, \tag{2.20}$$

Table 2.1: Values of $T[i]$ and $T[i]^{-1}$.

i	$T[i]$	$T[i]^{-1}$
1	0.707106781186547	1.414213562373095
2	0.840896415253715	1.189207115002721
3	0.917004043204671	1.090507732665258
4	0.957603280698574	1.044273782427414
5	0.978572062087700	1.021897148654117
6	0.989228013193975	1.010889286051701
7	0.994599423483633	1.005429901112803
8	0.997296056085470	1.002711275050203
9	0.998647112890970	1.001354719892108

$$h_{k,n}^{\text{lower}}(\mathbf{x}, i) = h_{k,n}^{\text{lower}}(\mathbf{x}, i-1)T[i]^{-\overline{\beta}_{k,n}^{[i]}}. \tag{2.21}$$

Since the inequalities

$$K_{k,n}\exp(-z_{k,n}) \leq h_{k,n}^{\text{upper}}(\mathbf{x}, i) \leq h_{k,n}^{\text{upper}}(\mathbf{x}, i-1), \tag{2.22}$$

$$h_{k,n}^{\text{lower}}(\mathbf{x}, i-1) \leq h_{k,n}^{\text{lower}}(\mathbf{x}, i) \leq K_{k,n}\exp(-z_{k,n}), \tag{2.23}$$

hold, Eqs. (2.16) and (2.17) become closer to the true evaluation $g_k(\mathbf{x})$ as i increases. Consequently, the intervals are refined as shown in Fig. 2.4. Although both of Eqs. (2.20) and (2.21) include multiplications, one refinement process requires only either of $T[i]$ or $T[i]^{-1}$ since $\overline{\beta}_{k,n}^{[i]}$ is the bit inverse of $\beta_{k,n}^{[i]}$. Both of $T[i]$ and $T[i]^{-1}$ can be calculated before and stored in look up tables as shown in Tab. 2.1.

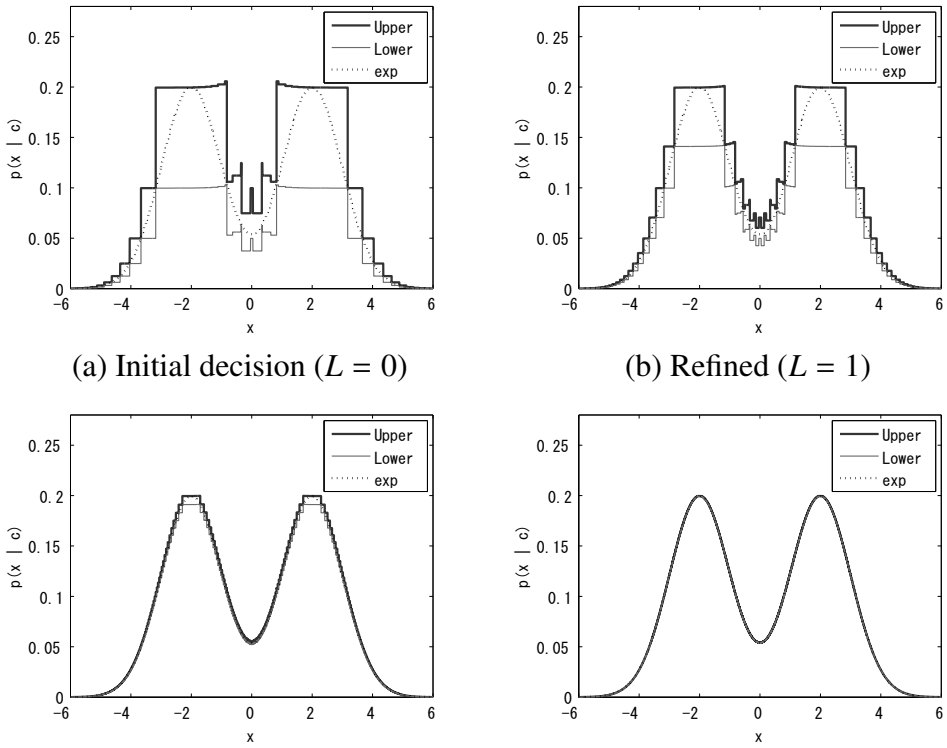
Figure 2.5 shows an example of interval calculations for a Gaussian mixture distribution. It is seen that the interval guarantees the true value and the interval is close to the true value as the number of refinement increases.

2.2.3 Termination process

The refined bounds approach to a true value on a principal by continuing the update operations. However, in an actual situation, computations must be completed in a finite bit accuracy. As an option, we provide a termination process. Let L be the maximum number of refinement process. Then, the average value of the bounds may be a good approximation of $g_k(\mathbf{x})$. The average is obtained by

$$g_k^{\text{pseudo}}(\mathbf{x}) = \frac{1}{2}\left\{g_k^{\text{lower}}(\mathbf{x}, L)\cdot T[L]^{-1} + g_k^{\text{upper}}(\mathbf{x}, L)\right\}. \tag{2.24}$$

Then, $g_k(\mathbf{x})$ in Eq. (2.5) can be replaced by $g_k^{\text{pseudo}}(\mathbf{x})$ for a pseudo decision. Note that $T[L]^{-1} \approx 1$ for a large L.



(a) Initial decision ($L = 0$) (b) Refined ($L = 1$)

(c) Refined ($L = 4$) (d) Refined ($L = 8$)

Figure 2.5: Refinement effect when a Gaussian mixture consists of $\frac{1}{2}\mathcal{N}(x| -2, 1)$ and $\frac{1}{2}\mathcal{N}(x|2, 1)$, where L is the number of refinement processes.

2.2.4 Proposed algorithm

Let us summerize the procedures described from 2.2.1 to 2.2.3, where we represent $2^{-i}x$ as $x \gg i$ and assume a two class case for the sake of convenience.

Step 1 Calculate Eqs. (2.6) and (2.7) for all $k \in \{0, 1\}$ and $n = 0, 1, \cdots, N_k - 1$. Then, obtain

$$h_{k,n}^{\text{upper}} = K_{k,n} \gg \lfloor z_{k,n} \log_2 e \rfloor,$$

$$h_{k,n}^{\text{lower}} = h_{k,n}^{\text{upper}} \gg 1,$$

$$\beta_{k,n} = \text{frac}(z_{k,n} \log_2 e),$$

and set $i = 0$, where $\text{frac}(x)$ is the fractional part of x.

Step 2 Calculate the following equations for all $k \in \{0, 1\}$:

$$g_k^{\text{upper}} = \sum_{n=0}^{N_k - 1} h_{k,n}^{\text{upper}}$$

$$g_k^{\text{upper}} = \begin{cases} g_k^{\text{upper}} \gg 1, i = 0 \\ \sum_{n=0}^{N_k-1} h_{k,n}^{\text{lower}}, i \geq 1 \end{cases}.$$

Step 3 If $g_0^{\text{upper}} \leq g_1^{\text{lower}}$, then decide on Class C_1 and quit. If $g_1^{\text{upper}} \leq g_0^{\text{lower}}$, then decide on Class C_0 and quit.

Step 4 Increment i as $i \leftarrow i + 1$. If i exceeds the upper limit L, then go to Step 6.

Step 5 Update the interval for all $k \in \{0, 1\}$ and $n = 0, 1, \cdots, N_k - 1$. If $\beta_{k,n}^{[i]} = 1$, then apply

$$h_{k,n}^{\text{upper}} \xleftarrow{\text{update}} h_{k,n}^{\text{upper}} \cdot T[i].$$

Otherwise, apply

$$h_{k,n}^{\text{lower}} \xleftarrow{\text{update}} h_{k,n}^{\text{lower}} \cdot T[i]^{-1}.$$

Return to Step 2.

Step 6 Decide the class with the pseudo-function

$$g_k^{\text{pseudo}}(\mathbf{x}) = \frac{1}{2}\left\{g_k^{\text{lower}}(\mathbf{x}, L) \cdot T[L]^{-1} + g_k^{\text{upper}}(\mathbf{x}, L)\right\}.$$

If $g_0^{\text{pseudo}} < g_1^{\text{pseudo}}$, then decide on Class C_1. Otherwise, decide on Class C_0. Finally, terminate the process.

2.3 Performance Evaluation

In this section, we verify the significance of the initial decision, and then evaluate the computational cost of the refinement process on both of DSP and FPGA.

2.3.1 Experiments with color extraction

More the initial decision completes the classification, the lower the computational cost becomes. Let us show some simulation results of the application to color extraction. The followings summerize the simulation procedure:

- Simulation 1
  - Claire, a sequence of size 144×176 in YUV format, was used.
  - A 2×1 feature vector $\mathbf{x}$ was defined for every pixel by the U and V component.
  - GMMs are assumed to the skin and non-skin class.
  - The first frame was used to train the classifier [1].
  - The 494-th frame was used as an observed picture.
- Simulation 2
  - Table tennis, a sequence of size 352×240 in YUV format, was used.
  - A 2×1 feature vector $\mathbf{x}$ was defined for every pixel by the U and V component.
  - GMMs are assumed to the uniform ware and other region class.
  - The 80-th frame was used to train the classifier.
  - The 90-th frame was used as an observed picture.

Figure 2.6 shows an initial decision result of Simulation 1, where the numbers of distributions are assumed to be two, three and four. It is observed that the initial decision process can almost complete the classification. The ratios of completing the classification for different numbers of distributions resulted in

- 99.629 % for 2 distributions ($N_0 = N_1 = 2$),
- 99.609 % for 3 distributions ($N_0 = N_1 = 3$),
- 99.562 % for 4 distributions ($N_0 = N_1 = 4$).

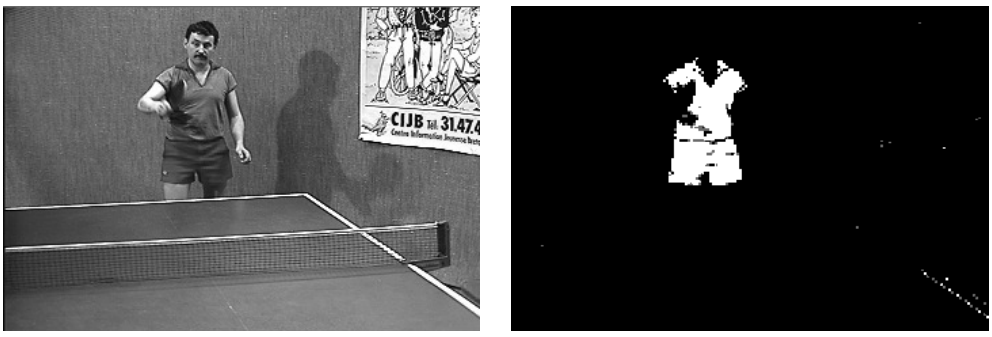
As well, we verified that the number of updates in which the decisions completed in the double precision were 7, 6 and 11 updates for 2, 3 and 4 distributions, respectively.

[1] EM algorithm was used [14, 15]

(a) Observation (b) Result with $N_0 = N_1 = 2$

(c) Result with $N_0 = N_1 = 3$ (d) Result with $N_0 = N_1 = 4$

Figure 2.6: Initial decision results in Claire, where the white, black and gray region express the skin, non-skin and undecided region.



(a) Observation (b) Result with $N_0 = N_1 = 2$

Figure 2.7: Initial decision results in Table tennis, where the white, black and gray region express the uniform, non-uniform and undecided region.

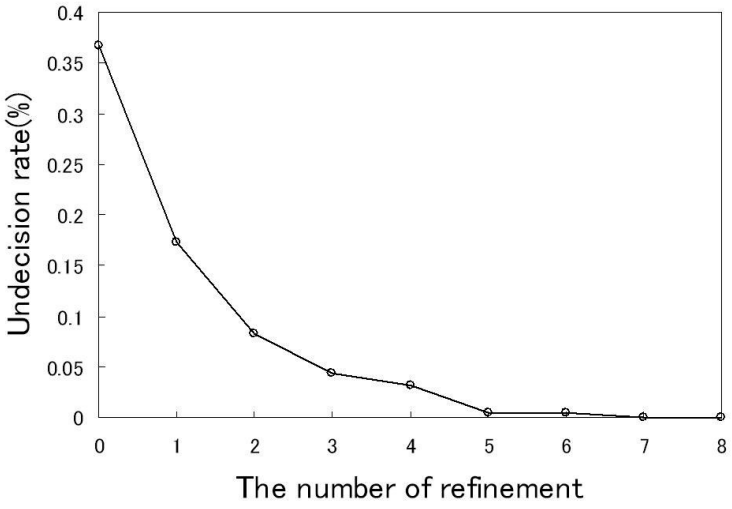


Figure 2.8: Undecision rate for each refinement stage in Claire, where $N_0 = N_1 = 2$.

Figure 2.7 shows an initial decision result of Simulation 2, where the number of distributions are assumed to be two. The ratios of completing the classification resulted in 99.670%.

These simulation results tell us that any precise evaluation is not necessary for the exponential function for over 99.5% of pixels.

Figure 2.8 shows an effect of the refinement process for Claire. It is confirmed that undecision rate is reduced as the number of refinement process incleases.

2.3.2 Computational cost on DSP

Let $r[i]$ be the ratio of progression to the i-th refinement process, where the relation $0 \leq r[i+1] \leq r[i] \leq 1$ holds. Denoting τ_{init} and τ_{refine} as the time required for the initial decision per feature vector in Steps 1, 2 and 3 and the time for a refinement process in Steps 5, 2 and 3, respectively, we obtain the average decision time per feature vector with L refinement processes as

$$\tau_{\text{ave}} = \tau_{\text{init}} + \sum_{i=1}^{L} r[i]\tau_{\text{refine}}, \tag{2.25}$$

where the termination in Step 6 is omitted for simplicity since it is optional and less expensive than the refinement process. If τ_{init} is much less than the time τ required with some existing exponential implementation, and if the progression ratios $r[i]$ are close to zero, then the acceleration can be expected since $\tau_{\text{ave}} \approx \tau_{\text{init}}$. For evaluating the computational cost, we implemented the skin-color extraction in Simulation 1 on TI DM6437 EVM board. The followings summarize the procedure:

- TI CCS ver. 3.3 was used as the compiler with the 'Speed Most Critical' and 'Function Level' option.

Table 2.2: Average computational time per feature vector for evaluating Eq.(2.4) on DM6437 EVM board, where $D = 2$ and $N_0 = N_1 = 2$. The quadratic form of Eq.(2.6) consumes around $5.0\mu s$ in floating-point calculations of single precision. $L = 0$ means without refinement. LPW was implemented by $a = 4, b = 8, f_1 = 0.0046$.

Type of constant $K_{k,n}$	Fixed-point	Float.-point
Std. (w. expf())	-	13.65 μs
Std. (w. exp())	-	25.8 μs
LPW.	-	1.59 μs
Intv. ($L = 0$)	1.34 μs	2.13 μs
Intv. ($L = 1$)	1.39 μs	2.13 μs
Intv. ($L = 2$ to $L = 8$)	1.40 μs	2.14 μs

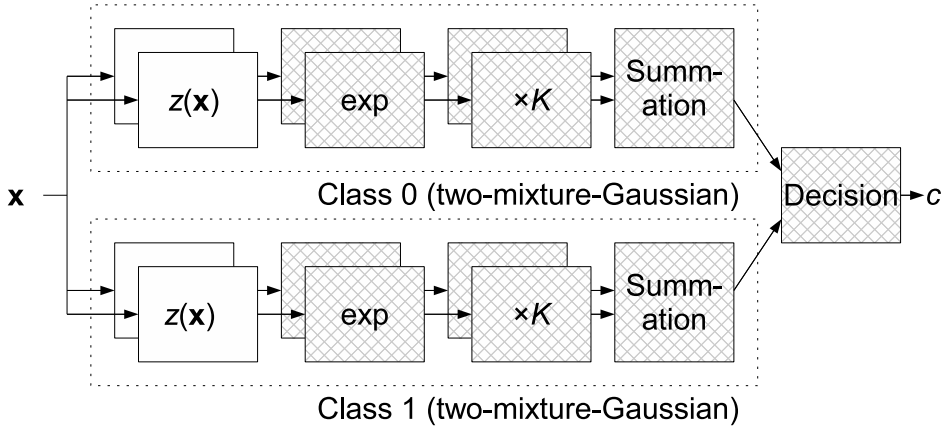
- Classifiers with standard exponential functions, linear piase-wize (LPW) approximation and ones with our proposal were implemented on the board [16].

- The 494-th frame of Claire was used as an observed picture, which was enlarged and converted into NTSC video so that it can be acquired by the board.

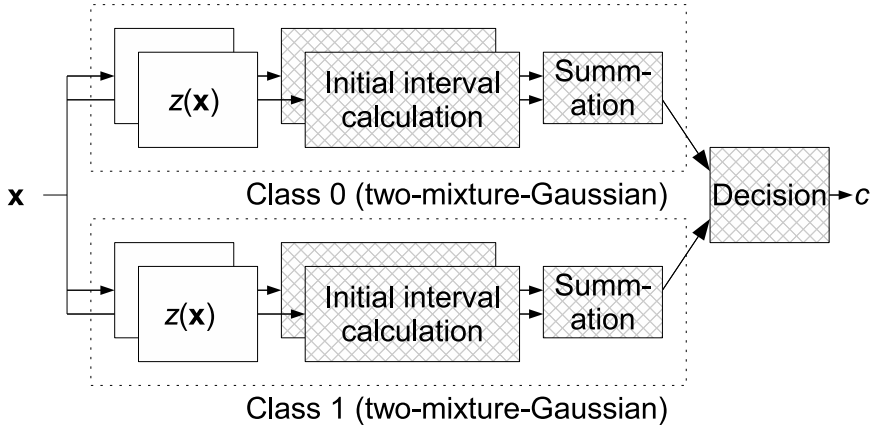
Table 1 summerizes the computational speed of the experimental results of the UV-based color extraction. The abbreviations 'Std.', 'LPW.' and 'Intv.' denote the classifiers with the standard functions, linear piece-wise approximation and the proposed algorithm with the interval calculation, respectively. As the standard exponential functions, we adopted 'expf()' of single precision and 'exp()' of double precision from the standard C library. For the proposal, the results only with the initial decision process ($L = 0$) and those with the refinement process up to 8 updates ($L = 8$) are given. Two types of representations, i.e. fixed- and floating-point one, for constant $K_{k,n}$ are also compared. Since the refinement process is rarely required, the overhead is negligible in both cases. We can verify that as the distributions of classes become close to each other, e.g. Kullback-Leibler divergence [14] becomes close to zero, the refinement ratios have a tendency to increase and may fail to accelerate the process. Although our proposed method is not universally efficient, it shows advantages of acceleration (or power saving) when the distributions are far from each other.

2.3.3 Computational cost on FPGA

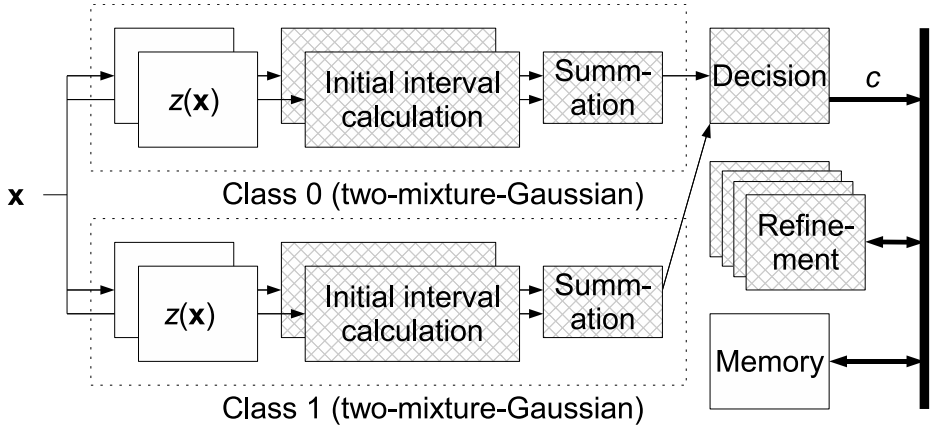
Let us verify the significance of our proposal in terms of speed and area on FPGA. A standard classifier with CORDIC, proposed classifier without the refinement, and one with the refinement were modeled by VHDL and synthesized onto FPGA. The followings compare these constructions in terms of the computational speed and area.



(a) Std. (w. CORDIC exp)

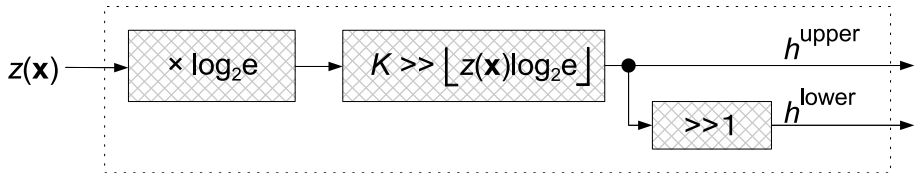


(b) Intv. (w.o. refine.)

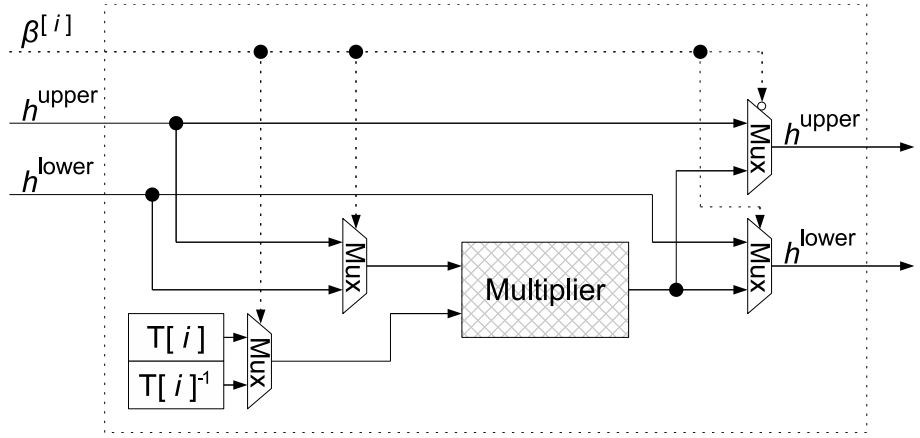


(c) Intv. (w. refine.)

Figure 2.9: Block diagrams of GMM-based classifiers.



(a) Initial decision block .



(b) Refinement process block .

Figure 2.10: Block diagrams of Initial decision and refinement process.

Figure 2.9 shows the block diagrams of the circuits adopted in this evaluation, where the hatched blocks denote the targets in this evaluation and Fig. 2.10(a), (b) shows the architectures of the initial decision and refinement process, respectively.

The CORDIC module was generated through Xilinx ISE LogiCore tool for the standard construction. The computational time and area of our proposal with the refinement process was estimated under the condition that all four of the refinement processes are executed in parallel and four constant multipliers are embedded in total.

The implementation targets the support of SXGA (112MHz, 9.0 ns) display and achives one decision per clock.

The followings summerize the evaluation procedure:

- TED TB-3S-3400DSP-IMG was used as an FPGA evaluation board,
- The board is equipped with Xilinx XC3SD3400DSP,
- Xilinx ISE 10.1 was selected as the development tool with the following settings:
  - Optimization: speed,
  - Architecture: fully pipelined,
  - Data type: 32-bit fixed-point.

Table 2.3: Computational time estimation by place-and-route timing report, where the percentages in the parentheses mean the ratio of Intv. to Std. with 32 bits.

	Clock rate (MHz)	Computational time(ns)
Std. (w. 32-bit exp)	112.03	8.93
Std. (w. 16-bit exp)	122.76 (109.58%)	8.15 (91.3%)
Intv. (w.o. refine.)	145.99 (130.31%)	6.85 (76.74%)
Intv. (w. refine.)	114.13 (101.87%)	8.76 (98.16%)

Table 2.4: Area estimation by map report, where the percentages in the parentheses mean the ratio of Intv. to Std. with 32 bits.

	LUTs	FFs
Std. (w. 32-bit exp)	21,998	22,403
Std. (w. 16-bit exp)	7,902 (35.92%)	8,011 (35.76%)
Intv. (w.o. refine.)	1,687 (7.67%)	1,668 (7.45%)
Intv. (w. 1-refine.)	6,008 (27.69%)	6,091 (26.82%)

Figure 2.11 shows the effect of the number of refinement process to the circuit area. The circuit area increases with the number of refinement process because the constant look-up table expands.

The place-and-route and map reports are shown in Tables 2.3 and 2.4. Note that the computational time of the proposed method with the refinement process was estimated under the condition that the total time consumed by the refinement processes in a frame is less than that of the initial decision process in a frame.

The proposals show superior performances in terms both of the speed and area to the conventional ones. Although the performances depend on the distributions of the inputs, some significant improvement can be expected by adopting the proposed method for other applications.

2.4 Summary

This chapter reduced the comparison of GMM to simple interval calculations and proposed an efficient Bayesian decision scheme. In order to verify the significance, some experimental results of the application to color extraction were shown. It was verified that the classification was almost completed with the simple initial decision process, and that the refinement process was able to improve the classification with low overhead in the computational cost. It is verified through the implementation on DSP and FPGA that the proposed algorithm is hardware-friendly.

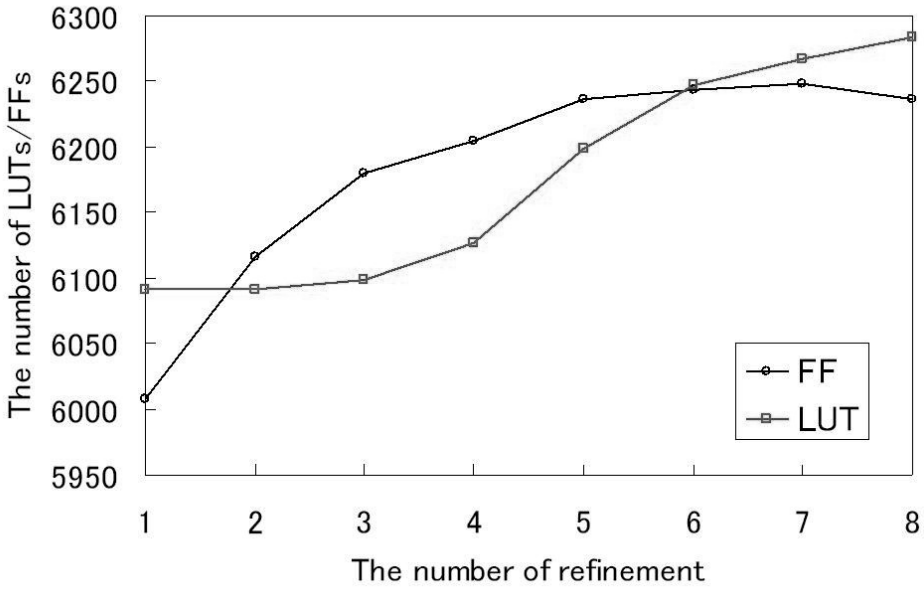


Figure 2.11: Relation between the number of LUTs/FFs and the number of refinement.

Chapter 3

Narrowed Initial Interval

This chapter proposes to improve the initial interval.

In Chapter 2, the interval is improved with refinement process. However, it is implemented as adoptive scheme. Therefore, the computational cost is increasing if the refinement is often processed.

Figure 3.1 illustrates undesirable GMMs. If one distribution is close to the other distribution, the refinement process is required over a wide range. It cause the increasing of the computational cost.

To avoid this problem, this chapter proposes an improvement of the initial interval. This chapter also uses GMM-based classification as the application of GMM.

3.1 Narrowed Initial Interval

The main issue of the interval calculation is the cost of the refinement process. The refinement process requires N multiplication when N-times refinements are executed. It remains possible that the cost of interval calculation becomes higher than the conventional exponent calculation when many refinement steps are required.

In this section, a fast refinement scheme is proposed. The scheme can reduce K multiplications and refinement steps by using newly introduced look-up tables.

3.1.1 Calculation of shift amount

The factor $K2^{-z(\mathbf{x})\log_2 e}$ can be rewritten as

$$K2^{-z(\mathbf{x})\log_2 e} = 2^{\log_2 K} \cdot 2^{-z(\mathbf{x})\log_2 e} = 2^{-\{-\log_2 K + z(\mathbf{x})\log_2 e\}}, \tag{3.1}$$

and we can define an shift amount $q_{k,n}$ by

$$q_{k,n}(\mathbf{x}) = \lfloor -\log_2 K_{k,n} + z_{k,n}(\mathbf{x})\log_2 e \rfloor. \tag{3.2}$$

If $\log_2 K_{k,n} > z_{k,n}(\mathbf{x})\log_2 e$, the shift amount $q_{k,n}(\mathbf{x})$ become greater than 1. It means that left bit-shift is required in the fixed-point implementation as well as

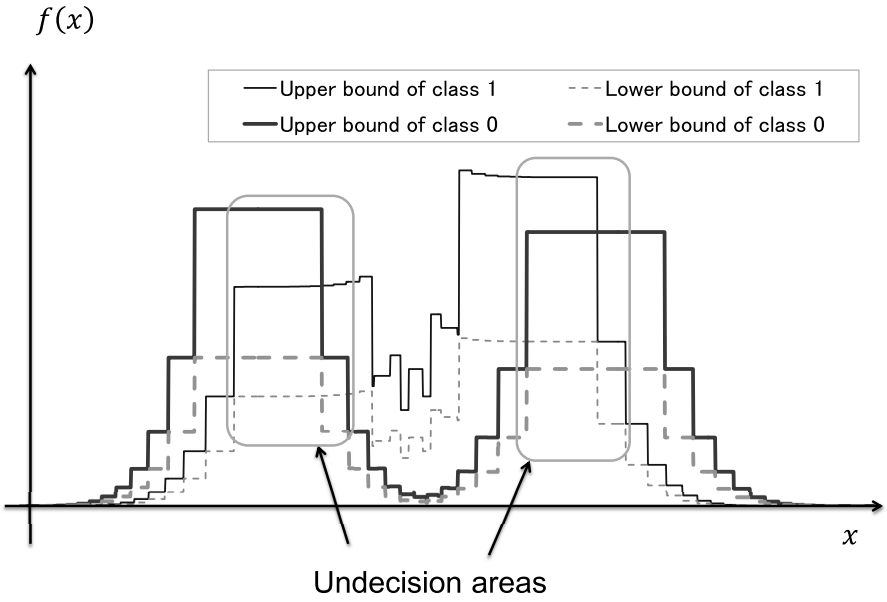


Figure 3.1: An example of intervals with undesirable GMMs. Thin solid line and thin dashed line show the interval bound of class 1 and bold solid line and bold dashed line show the interval bound of class 0. Rounded rectangle shows the undecision areas. If one distribution is close to the other distribution, the refinement process is required over a wide range.

right bit shift. It makes the fixed-point implementation complex. To avoid the left bit shift, let us normalize $K_{k,n}$ as

$$\overline{K}_{k,n} = K_{k,n}/\max\{K_{k,n}\}. \tag{3.3}$$

Note that $\log_2 \overline{K}_{k,n}$ is not positive because $\overline{K}_{k,n}$ is less than or equal to 1. Equation (3.3) is based on the fact that GMM classification with Bayesian decision rule can be written as

$$if \max_{i=0,\cdots,N-1} \left\{ \frac{P(C_i)\mathcal{M}(\mathbf{x}|\mathbf{\Theta}_i)}{U} \right\} = \frac{P(C_n)\mathcal{M}(\mathbf{x}|\mathbf{\Theta}_n)}{U}$$

$$\Rightarrow \mathbf{x} \in \text{class } n, \tag{3.4}$$

where U is a positive value.

From Eq. (3.3), the shift amount $q_{k,n}(\mathbf{x})$ is rewritten as

$$q_{k,n}(\mathbf{x}) = \lfloor -\log_2 \overline{K}_{k,n} + z_{k,n}(\mathbf{x}) \log_2 e \rfloor. \tag{3.5}$$

Equation (3.5) can be realized only by some right bit shifts in the fixed-point implementation.

3.1.2 Look-up table for multiplierless interval calculation

The look-up table of our proposed multiplierless scheme is different from the original one.

In the multiplierless scheme, the fractional part β of the exponential part is given by

$$\beta = -\log_2 \overline{K} + z(\mathbf{x}) \log_2 e - \lfloor -\log_2 \overline{K} + z(\mathbf{x}) \log_2 e \rfloor, \tag{3.6}$$

and a function $s(\beta)$ is defined as

$$s(\beta) = 2^{-\beta}. \tag{3.7}$$

In the multiplierless scheme, function $s(\beta)$ is realized by look-up table .

3.1.3 Refinement process with bit shift and look-up table

The shift amount and the table from Sec. 3.1.1 and 3.1.2 make it possible to reduce the multipliers.

With Eqs.(3.5) and (3.7), the refined interval operation is realized by

$$g_k^{\text{upper}}(\mathbf{x}) = \sum_{n=0}^{N-1} h_{k,n}^{\text{upper}}(\mathbf{x}), \tag{3.8}$$

$$g_k^{\text{lower}}(\mathbf{x}) = \sum_{n=0}^{N-1} h_{k,n}^{\text{lower}}(\mathbf{x}), \tag{3.9}$$

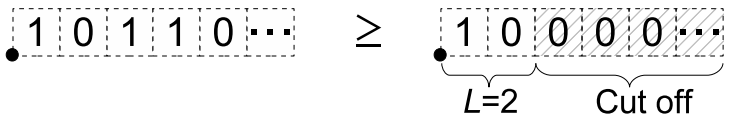


Figure 3.2: Relation between the true value (left-hand side) and round-down value (right-hand side). Round-down value becomes 0.5.

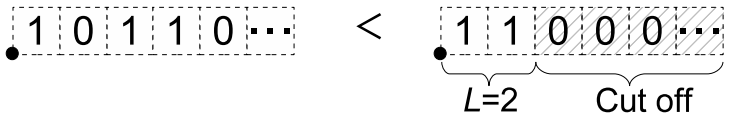


Figure 3.3: Relation between the true value (left-hand side) and round-up value (right-hand side). Round-up value becomes 0.75.

where

$$h_{k,n}^{\text{upper}}(\mathbf{x}) = s(\beta_{k,n}) \gg q_{k,n}(\mathbf{x}), \tag{3.10}$$

and

$$h_{k,n}^{\text{lower}}(\mathbf{x}) = \begin{cases} s(0) \gg (q_{k,n}(\mathbf{x})+1) & (\beta_{k,n} = 1-2^{-L}) \\ s(\beta_{k,n}+2^{-L}) \gg q_{k,n}(\mathbf{x}) & (otherwise) \end{cases}, \tag{3.11}$$

where $\gg$ is the barrel shifter. Note that there is no multiplier in Eqs. (3.10) and (3.11).

Equation (3.11) is based on the following inequation:

$$\sum_{l=1}^{L} \beta_l \le \sum_{l=1}^{\infty} \beta_l < \left(\sum_{l=1}^{L} \beta_l \right) + 2^{-L}, \tag{3.12}$$

where the l-th bit of β is expressed by β_l, $\sum_{l=1}^{L} \beta_l$ is the L-bit round-down of β and $\sum_{l=1}^{\infty} \beta_l$ is the true value of β. Figures 3.2 and 3.3 show some examples for $L = 2$ and true value 0.6875.

3.1.4 Decision rule in the proposed refinement scheme

The decision rule is given by

$$\begin{aligned} if\ \{g_k^{\text{lower}}(\mathbf{x}) - g_i^{\text{upper}}(\mathbf{x})\} > 0\ for\ all\ i \\ \Rightarrow \mathbf{x} \in \text{class } k, \end{aligned} \tag{3.13}$$

where $k, i = 0, 1, \cdots, N-1$.

3.2 Performance Evaluation on FPGA

In this section, in order to verify the significance of the novel refinement scheme, let us evaluate the circuit size of GMM-based classifier with the interval calculation in FPGA is evaluated.

3.2.1 Implementation condition

In the followings, we summarize the design condition for a GMM-based classifier with the interval calculation

- Target FPGA: Xilinx XC3SD3400
- Development environment: Xilinx ISE 10.1
- Optimization: speed
- Data type: 32-bit and 64-bit fixed-point
- The number of classes: 2
- The number of distributions: 2
- Embedded RAMs and multipliers are not used

The original refinement scheme was estimated by adding four multipliers to the initial decision process. The multipliers were generated by Xilinx Core Generator and they were configured as parallel architecture.

Figures 3.4 and 3.5 show the block diagrams of the circuits in this evaluation, where the hatched blocks denote the target in this evaluation.

3.2.2 Synthesis report

Tables 3.1-3.4 show the synthesis results of the number of flip-flops, look-up tables (as elements of FPGA), max clock period, throughput, and clock latency. The values in parentheses show the ratio of the improvement from the proposed scheme to the original scheme. Symbols that + or − on Table 3.3 and 3.4 mean uncertainty of the estimated values because the original refinement process is adaptive scheme, and the value shows the best case where only a few refinement processes are required in the interval calculation.

From these results, it was confirmed that the multiplierless refinement scheme for the interval calculation can reduce the circuit area under 52.2% and can accelerate the throughput over 117.6%.

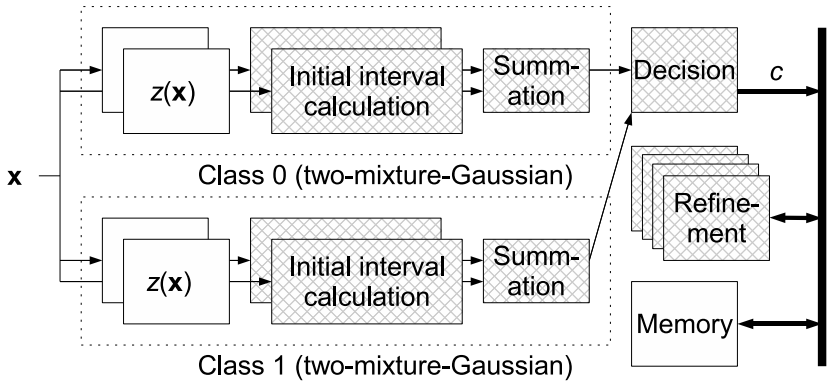


Figure 3.4: Original interval calculation classifier.

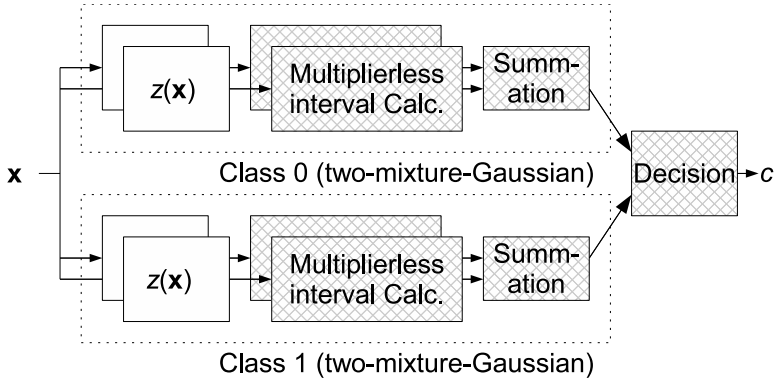


Figure 3.5: Proposed interval calculation classifier.

3.3 Summary

In this chapter, a method to improve the initial interval was proposed for GMM and it was applied for GMM-based classification. The proposed scheme consists of scaling for the exponential part and shifting of pre-calculated values of the fractional part. As a result, the initial interval could be narrowed. By the experimental implementation on FPGA, it was confirmed that the proposed method reduced the circuit area under 52.2% and accelerated the throughput over 117.6%.

Table 3.1: Results of logic synthesis for 32-bit implementation.

	FFs	LUTs	Max clock period (MHz)
Prop. 1-bit	2690(42.2%)	2423(40.9%)	180.343(117.6%)
Prop. 2-bit	3038(47.6%)	2778(46.9%)	180.343(117.6%)
Prop. 3-bit	3274(51.3%)	3031(51.1%)	180.343(117.6%)
Prop. 4-bit	3302(51.8%)	3095(52.2%)	180.343(117.6%)
Orig. refine.	6380	5928	153.374

Table 3.2: Results of logic synthesis for 64-bit implementation.

	FFs	LUTs	Max clock period (MHz)
Prop. 1-bit	4385(22.2%)	4043(21.0%)	178.253(171.84%)
Prop. 2-bit	5078(25.7%)	4720(24.5%)	178.253(171.84%)
Prop. 3-bit	5547(28.0%)	5188(27.0%)	178.253(171.84%)
Prop. 4-bit	5615(28.39%)	5320(27.7%)	178.253(171.84%)
Orig. refine.	19779	19232	103.734

Table 3.3: Results of throughput and clock latency for 32-bit implementation.

	Throughput($\times 10^6$)	Latency
Prop. 1-bit	180.343(117.6+%)	12
Prop. 2-bit	180.343(117.6+%)	12
Prop. 3-bit	180.343(117.6+%)	12
Prop. 4-bit	180.343(117.6+%)	12
Orig. refine.	153.374-	9+

Table 3.4: Results of throughput and clock latency for 64-bit implementation.

	Throughput($\times 10^6$)	Latency
Prop. 1-bit	178.253(171.84+%)	13
Prop. 2-bit	178.253(171.84+%)	13
Prop. 3-bit	178.253(171.84+%)	13
Prop. 4-bit	178.253(171.84+%)	13
Orig. refine.	103.734-	11+

Chapter 4

LUT-Based GMM

This chapter proposes LUT-based calculation for GMM. In Chapters 2 and 3, the interval representation was discussed. The interval can be interpret as the upper and lower band of the approximation error. Therefore, this chapter proposes the computation of the LUT for exponential function based on Chapters 2 and 3, and it is shown that the values in the LUT can be computed with flexibility.

The LUT-based exponential function is applied for Expectation Maximization (EM) algorithm for GMM and evaluated in this chapter.

4.1 Review of EM Algorithm for GMM

In this section, the EM algorithm for the GMM is reviewed and discussed the computational complexity. Especially, this section focuses on expectation step (E-step) of the EM algorithm. For more information on the EM algorithm, refer to [17, 14].

4.1.1 E-step

The EM algorithm consists of two principal processes. First, the E-step calculates probabilities, which are called responsibilities. Then, the maximization-step (M-step) updates the parameters of GMM by using these responsibilities. These two steps are repeated until the parameters converge. In the followings, let us show the equation of the E-step where the notations are based on [14].

In the E-step, we compute the responsibilities $\gamma_{k,n}$, where k is the distribution number and n is the data index. Suppose that we observe input vectors $\{\mathbf{x}_n\}_{n=0}^{N-1}$, where $\mathbf{x}_n \in \mathbb{R}^{D\times 1}$ and N is the number of data points. The responsibility $\gamma_{k,n}$ of the k-th distribution for the n-th input vector $\mathbf{x}_n$ is given by

$$\gamma_{k,n} = \frac{\alpha_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=0}^{K-1} \alpha_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}, \tag{4.1}$$

where K is the number of Gaussian distributions, α_k is the mixture ratio of the k-th distribution, and $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the mean vector and covariance matrix of the k-th

distribution, respectively, where $\boldsymbol{\mu}_k \in \mathbb{R}^{D\times 1}$ and $\boldsymbol{\Sigma}_k \in \mathbb{R}^{D\times D}$. $\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is the k-th multivariate Gaussian distribution given as

$$\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = C_k \cdot \exp(-y_{k,n}), \tag{4.2}$$

where

$$C_k = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_k|^{1/2}}, \tag{4.3}$$

and

$$y_{k,n} = \frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k). \tag{4.4}$$

Note that C_k and $y_{k,n}$ are non-negative scalar values.

For the M-step, refer to the article [14].

4.1.2 Computation of EM algorithm for GMM

In the followings, we present our approach for reducing the cost associated with computing the E-step. Let us decompose the computational procedures in Eq. (4.1) as follows:

1. $y_{k,n} = \frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k)$,
2. $u_{k,n} = \exp(-y_{k,n})$,
3. $C_k = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_k|^{1/2}}$,
4. $t_{k,n} = \alpha_k C_k u_{k,n}$,
5. $s_n = \sum_{k=0}^{K-1} t_{k,n}$,
6. $\bar{s}_n = 1/s_n$,
7. $\gamma_{k,n} = t_{k,n} \cdot \bar{s}_n$.

Operations 1, 2, and 3 presented above are prone to bottlenecks due to matrix products, exponential functions, and determinant operations. Even when the number of data n is large, the computational cost of C_k is not significant because C_k is independent of n. Thus, we need to reduce the computational cost of Operations 1 and 2. The most significant operation in the E-step is listed in Eq. (4.2).

The aim of this work is to reduce the operations in Eq. (4.2). The exponential function in operation 2 of the E-step can be approximated by taking into account Eq. (4.1). In the next section, we propose a technique for approximating the exponential function.

4.2 EM Algorithm with Look-Up-Table-Based Exponential Function

In this section, we propose an approximation method for the exponential function used by the EM algorithm that estimates the GMM parameters. The approximation method reduces computational cost by taking Eq. (4.1) into account.

The idea of transforming non-linear function into look-up table (LUT) was used in various studies. For example, Fiori used the idea for neural network [18, 19, 20].

This work approximates the exponential function by using a bit-shift and a LUT. First, the exponential function is represented by powers of two. Then, the powers of two are approximated using a bit-shift and a LUT. Furthermore, to simplify the implementation, the LUT is scaled by a constant coefficient.

4.2.1 Look-Up-Table-Based Exponential Function

The exponential function can be expressed as a power of two as

$$\exp(-z) = 2^{-z \cdot \log_2 e}, \tag{4.5}$$

where z is a variable and e is Napier's number. The right-hand side of Eq. (4.5) can be separated into two components

$$2^{-z \cdot \log_2 e} = 2^{-(\lfloor z \cdot \log_2 e \rfloor + \beta)} = 2^{-\lfloor z \cdot \log_2 e \rfloor} \cdot 2^{-\beta}, \tag{4.6}$$

where $\lfloor x \rfloor$ represents the integer part of x and $\beta = z \cdot \log_2 e - \lfloor z \cdot \log_2 e \rfloor$, i.e., the fractional part of $z \cdot \log_2 e$.

From Eqs. (4.5) and (4.6) we conclude that in the binary digit system, exponential function can be realized by a bit-shift of $2^{-\beta}$. However, computing $2^{-\beta}$ still remains an issue. In particular, the power of two can be computed using the Taylor series expansion. Since the later approach is much simpler than the former, we propose introducing a LUT that contains approximated values of $2^{-\beta}$.

To construct a LUT with a finite number of contents, we use a bit string $\hat{\beta}$ that is the L-bit approximation of β. By using the bit string $\hat{\beta}$, the function $2^{-\beta}$ can be approximated by:

$$2^{-\beta} \approx 2^{-\hat{\beta}} = 2^{-\sum_{i=1}^{L} 2^{-i} \cdot \hat{\beta}^{[i]}}, \tag{4.7}$$

where, $\hat{\beta}^{[i]} \in \{0, 1\}$ is the i-th bit of $\hat{\beta}$. Note that the most significant bit and least significant bit of $\hat{\beta}$ are $\hat{\beta}^{[1]}$ and $\hat{\beta}^{[L]}$, respectively. By applying Eq. (4.7), the entries of the LUT $T[\hat{\beta}]$ are obtained by

$$T[\hat{\beta}] = 2^{-\sum_{i=1}^{L} 2^{-i} \cdot \hat{\beta}^{[i]}}. \tag{4.8}$$

Since LUT does not depend on any data, it can be computed off-line. In Tab. 4.1, we present the values of the LUT for $L = 2$.

Using Eqs. (4.6) and (4.8), the exponential function is approximated by

$$\exp(-z) \approx 2^{-\lfloor z \log_2 e \rfloor} \cdot T[\hat{\beta}]. \tag{4.9}$$

Equation (4.9) indicates that the exponential function can be computed by shifting the bits of an entry of the LUT.

Table 4.1: Values of the LUT $T[\hat{\beta}]$ for $L = 2$. The symbol $(\ )_2$ indicates that the bit string $\hat{\beta}$ is represented in binary form.

$\hat{\beta}$	$T[\hat{\beta}]$
$(00)_2$	$1.000000\cdots$
$(01)_2$	$0.840896\cdots$
$(10)_2$	$0.707107\cdots$
$(11)_2$	$0.594604\cdots$

4.2.2 Scale adjustment for look-up table

According to Eq. (4.8), the range of the entries of the LUT is $(0.5, 1.0]$. Two issues must be considered with respect to this range. One is the complexity of a floating-point representation, such as the IEEE 754 floating-point format. In this format, values within the range $(0.5, 1.0)$ change the mantissa part, while the value 1.0 changes the exponential part. Therefore, the value 1.0 requires exception handling. The second issue is that the values of the LUT require an additional bit because in fixed-point representation, value 1.0 requires one additional bit than the values in the range (0.5, 1.0). Therefore, the ranges $[0.5, 1.0)$ and $(0.5, 1.0)$ are preferable to range $(0.5, 1.0]$.

For transforming the values of LUT into preferable range, let us consider a Gaussian distribution scaled by a constant factor. In the E-step, scaling does not affect the result of $\gamma_{k,n}$. Equation (4.1) can be represented by

$$\gamma_{k,n} = \frac{\alpha_k \delta \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=0}^{K-1} \alpha_j \delta \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}, \tag{4.10}$$

where δ is a non-zero constant. It is clear that the result of Eq. (4.10) is identical to that of Eq. (4.1). Hence, for Gaussian distributions, the scaling operation has no affect on the results of the E-step.

Scaling can be used to modify the look-up-table-based exponential function. The scaled Gaussian distribution is approximated by

$$\begin{aligned} \delta \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) &= \delta C_k \exp\{-y_{k,n}\} \\ &\approx \delta C_k \cdot 2^{-\lfloor y_{k,n} \log_2 e \rfloor} T[\hat{\beta}_{k,n}] \\ &= C_k \cdot 2^{-\lfloor y_{k,n} \log_2 e \rfloor} \hat{T}[\hat{\beta}_{k,n}], \end{aligned} \tag{4.11}$$

where $\hat{T}[\hat{\beta}_{k,n}] = \delta T[\hat{\beta}_{k,n}]$. Hence, the LUT can be scaled by δ.

4.2.3 Scaling using the weighted average of LUT entries

Next, we address the problem of identifying δ that scales the LUT within the desired range. In order to obtain an appropriate constant, we consider generating the

Table 4.2: Values of the LUT $\hat{T}[\hat{\beta}]$ obtained by the weighted-average method for $L = 2$ and $a_0 = a_1 = 0.5$.

$\hat{\beta}$	$\hat{T}[\hat{\beta}]$
$(00)_2$	$0.920448\cdots$
$(01)_2$	$0.774002\cdots$
$(10)_2$	$0.650855\cdots$
$(11)_2$	$0.547302\cdots$

LUT entries by linearly combining neighboring entries. The next entry of $T[\hat{\beta}]$ is represented by $T[\hat{\beta} + 2^{-L}]$. Using a weighted average approach, the scaled value $\hat{T}[\hat{\beta}]$ is calculated by

$$\begin{aligned}\hat{T}[\hat{\beta}] &= a_0 T[\hat{\beta}] + a_1 T[\hat{\beta} + 2^{-L}] \\ &= a_0 \cdot 2^{-\sum_{i=1}^{L} 2^{-i}\cdot\hat{\beta}^{[i]}} + a_1 \cdot 2^{-\{(\sum_{i=1}^{L} 2^{-i}\cdot\hat{\beta}^{[i]})+2^{-L}\}} \\ &= 2^{-\sum_{i=1}^{L} 2^{-i}\cdot\hat{\beta}^{[i]}} \left(a_0 + a_1 \cdot 2^{-2^{-L}}\right) \\ &= T[\hat{\beta}]\left(a_0 + a_1 \cdot 2^{-2^{-L}}\right), \end{aligned} \tag{4.12}$$

where a_0 and a_1 are weights, $a_0 + a_1 = 1$ and $a_0, a_1 \geq 0$. Equation (4.12) converts the range of the LUT into the range $[2^{-\sum_{i=1}^{L} 2^{-i}} \cdot (a_0 + a_1 \cdot 2^{-2^{-L}}), a_0 + a_1 \cdot 2^{-2^{-L}}]$, where $\sum_{i=1}^{L} 2^{-i}$ indicates that all digits of $\hat{\beta}$ are equal to one, i.e., $\hat{\beta} = (11\cdots11)_2$. Note that because $\left(a_0 + a_1 \cdot 2^{-2^{-L}}\right)$ is independent of $\hat{\beta}$, it becomes a constant. Thus, the constant $\left(a_0 + a_1 \cdot 2^{-2^{-L}}\right)$ can be used as the scaling factor δ. In Tab. 4.2, we present the values of the LUT for $L = 2$ and $a_0 = a_1 = 0.5$.

4.3 Performance Evaluation

To validate the effectiveness of the proposed method, we generate simulation results and evaluate the precision of the estimated parameters and computational time of the EM algorithm.

Random numbers from two-component mixture of Gaussian distributions are generated as follows:

- The Mersenne twister method is used to generate uniform random numbers [21].
- The Box-Muller method is used to generate Gaussian random numbers [22].
- The number of data points is 100,000.

The parameters of the distributions are summarized in Tab. 4.3.

Next, we use the EM algorithm to estimate the parameters. We apply the E-step and M-step 30 times in a loop. The initial parameters used for the EM algorithm are

Table 4.3: Parameters of a mixture Gaussian distribution for generating normal random numbers, where α is the mixture ratio, μ is the mean, and σ is the variance. The step sizes of μ_1 and Σ_1 are 0.2.

Params.	Dist. 0	Dist. 1
α	0.5	0.5
μ	0.0	From 1.0 to 5.0
Σ	1.0	From 0.2 to 2.0

Table 4.4: Initial values used in the EM algorithm.

Params.	Dist. 0	Dist. 1
α^{init}	0.5	0.5
μ^{init}	-0.5	μ_1+0.5
Σ^{init}	1.0	1.0

summarized in Tab. 4.4. Note that initial mean values are different from the original values.

The exponential function in the EM algorithm was implemented using the proposed method, the Taylor expansion, and the exp() function in the standard C library. Moreover, during the simulation, we adopted the IEEE 754 double precision format.

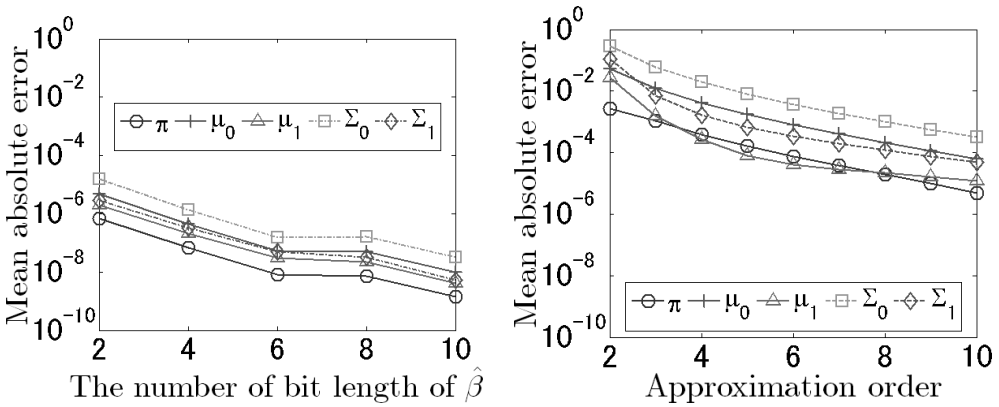
The evaluation program was implemented using the C programming language. The specifications of the software development environment and hardware environment used are:

- OS: MS Windows 7 (64 bit edition)
- Development environment: MS Visual Studio 2010
- SDK: MS Windows SDK for Windows 7
- Optimization flags: /Ox and /arch:SSE2
- CPU: Intel Core 2 duo E8500 (3.16 GHz)
- Memory: 8 GB dual channel DDR2 SDRAM (PC2-6400)

The simulation results were evaluated on a single core.

4.3.1 Precision of parameter estimation

In this section, we present representative simulation results. The simulation results were evaluated in terms of the mean absolute error (MAE). The results obtained by the exp() function are considered as the true values.



(a) Proposed method. (b) Taylor expansion.

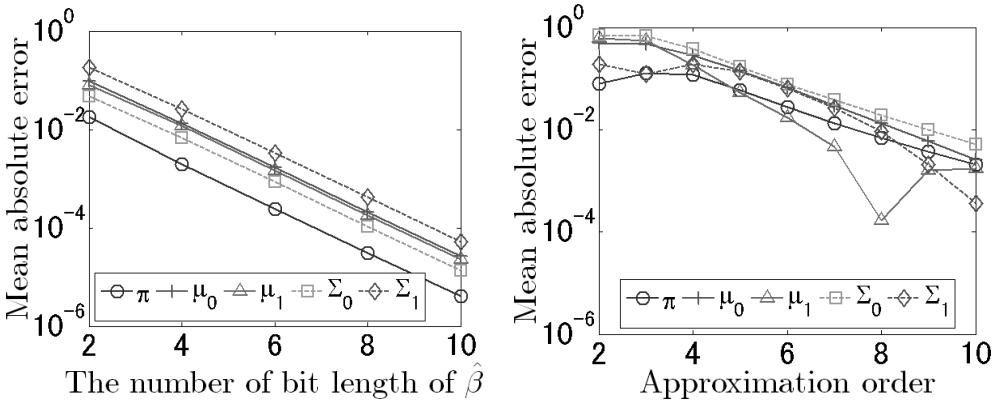
Figure 4.1: Simulation results obtained with $\mu_1 = 5.0$ and $\Sigma_1 = 0.2$. The vertical axis represents the mean absolute error. The horizontal axis in (a) represents the bit-length of $\hat{\beta}$, while the horizontal axis in (b) is approximation order. The averages obtained by exp() were $\pi_0 = 0.500, \pi_1 = 0.500, \mu_0 = 6.204 \times 10^{-4}, \mu_1 = 5.000, \Sigma_0 = 1.000$, and $\Sigma_1 = 0.200$.

The first simulation was performed with $\mu_1 = 5.0$ and $\Sigma_1 = 0.2$. The second was performed with $\mu_1 = 1.0$ and $\Sigma_1 = 2.0$. In Fig. 4.1, we present simulation results obtained for $\mu_1 = 5.0$ and $\Sigma_1 = 0.2$. The results obtained from the proposed method are more precise than those obtained from the Taylor expansion. Figure 4.2 shows the results for $\mu_1 = 5.0$ and $\Sigma_1 = 0.2$. For this parameter combination, the two distributions are close and overlap. In this case, it is difficult to estimate the parameters. The results obtained by the proposed method are comparable to those of the Taylor expansion.

4.3.2 Computational speed

Next, let us discuss the computational time of E-step using the proposed method, Taylor expansion, and the standard exp() function. We recorded the computational times of 30 loops of E-step and M-step by obtaining 100 measurements using 100 different random number seeds, and computed the average time.

The computational time of the E-step using the standard exp() function and the direct implementation of M-step was 14.26 [ms] and 1.26 [ms], respectively. These results demonstrate that for signal variables, the computational time of E-step is dominant in the EM algorithm. Figure 4.3 shows the computational time of the E-step using the proposed method and the Taylor expansion. The proposed method took less than 6.54 [ms], while the Taylor expansion took over 7.01 [ms]. Specifically, the Taylor expansion with higher-order approximation required over 10.73 [ms]. By comparing the proposed method with 10-bits and the Taylor expansion with the 10-th order approximation, we observed that the proposed method reduces the computational time by 45.62% while achieving more precise estimation results.



(a) Proposed method. (b) Taylor expansion.

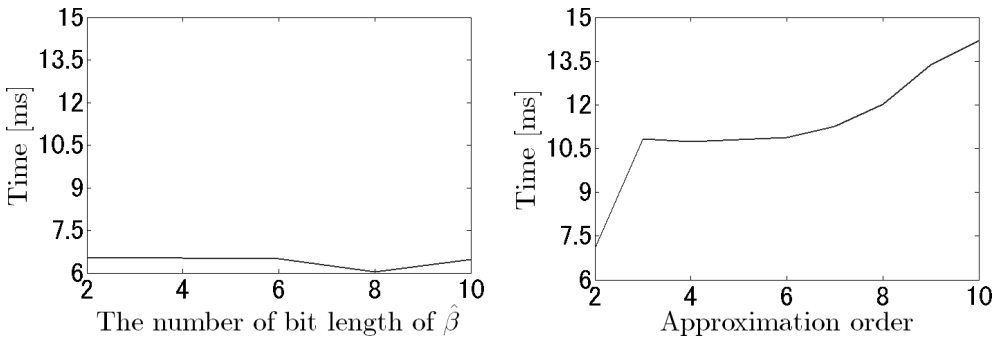
Figure 4.2: Simulation results obtained for $\mu_1 = 1.0$ and $\Sigma_1 = 2.0$. The vertical axis represents the mean absolute error. The horizontal axis in (a) represents the bit-length of $\hat{\beta}$, while the horizontal axis in (b) is the approximation order. The averages obtained by exp() were $\pi_0 = 0.549, \pi_1 = 0.451, \mu_0 = 2.046 \times 10^{-3}, \mu_1 = 1.106, \Sigma_0 = 1.045$, and $\Sigma_1 = 1.934$.

Similarly, by comparing the proposed method with the standard exp() function, we observe that the proposed method reduces computational time by more than 45.86%.

4.4 Summary

This chapter discussed the value of LUT for the exponential function and applied the EM algorithm to estimate the parameters in a GMM. First, the exponential function in the E-step was converted into a power of two. Then, the exponential function was converted using a LUT. The LUT was scaled using a weighted-average technique and the computational cost was reduced.

Through simulation results, it was demonstrated that the mean absolute error and computational time were reduced compared to the Taylor expansion. The proposed method was also shown to maintain high precision.



(a) Proposed method. (b) Taylor expansion.

Figure 4.3: Computational time results. The vertical axis shows the computational time. The horizontal axis in (a) is the bit-length of $\hat{\beta}$, and the horizontal axis in (b) is the approximation order.

Chapter 5

Conclusions

5.1 Summary

This study proposed efficient arithmetics for the GMM. The contributions of this study are two approach for GMM approximation focus on exponential function. One is the interval calculation for the GMM. While well-known approximation methods, such as fixed-point implementation, are able to reduce the computational cost, the computational accuracy is dropped. In contrast, the interval approach was balancing the computational cost with the computational accuracy. The other is LUT-based GMM. It was shown that the values in LUT could be scaled with a scale factor under certain conditions. The scale factor provided flexibility for the value in LUT. As a result, the bit length of LUT could be reduced.

The summarizations of each chapters are as follows:

Chapter 2

In chapter 2, the interval calculation method was applied for the GMM-based classification with Bayesian decision rule. This chapter proposed three processes for classification. In the initial decision process, the coarse interval is calculated. It was confirmed that the initial decision process could be classified almost inputs. In the refinement process, the interval is narrowed adoptively. In the termination process, the forcible decision rule was proposed in which case the error classification is allowed. The computational cost and accuracy of these methods were evaluated with the computational time on DSP, the circuit areas on FPGA, and the rate of undecision on PC simulation.

Chapter 3

In Chapter 3, the initial interval was improved. It was indicated the possibility that the approximation accuracy is not satisfied only by the initial interval in Chapter 2. While the interval was narrowed adoptively in Chapter 2, this chapter improved the interval at a time. As a result, the narrowed interval without the refinement process

could be achieved under certain conditions. This method was evaluated with the circuit areas on FPGA.

Chapter 4

In Chapter 4, it was shown that the values in LUT for the approximation of exponential function could be scaled under certain conditions. As a result, bit length of the LUT could be reduced. The significance of this proposal was confirmed with the EM algorithm for GMM parameter estimation.

5.2 Open Problems

There remain some problems for the proposed approaches.

First, this study will be applied other models such as mixture of exponential families. For example, hidden Markov model (HMM) also involves the computation of the exponential function. Therefore, it is desired the evaluation of other models.

Second is the LUT size in the narrowed initial interval. The LUT size will be too large if the number of bit length is large. For the LUT size problem, the idea of LUT separation method, such as Half-square multiplier will be helpful technique[23].

Finally, the quadratic form in GMM is not reduced enough the computational complexity. Takahashi, et al. proposed reduced adder graph algorithm for the quadratic form in GMM [24]. It is indicated the possibility that the method can be reduce the areas on FPGA. This method should be evaluated in various situation.

Appendix A: Derivation of Eq. (2.15)

The following relation is used for deriving Eq (2.15):

$$\begin{aligned} 1-\beta &= \left(\sum_{i=1}^{L} 2^{-i} + 2^{-L}\right) - \beta \\ &= \left(\sum_{i=1}^{L} 2^{-i} - \beta\right) + 2^{-L} \\ &= \sum_{i=1}^{L} \left(1 - \beta^{[i]}\right) 2^{-i} + 2^{-L} \\ &= \sum_{i=1}^{L} \bar{\beta}^{[i]} 2^{-i} + 2^{-L} \end{aligned} \tag{5.1}$$

List of Figures

1.1 Organization 3

2.1 An example of GMM-based classification. 6
2.2 Relation of the inequality in Eq. (2.8). 9
2.3 Case that the interval is used for classification. 9
2.4 Case that the interval computation is insufficient. 10
2.5 Examples of refinement effect. 13
2.6 Initial decision results in Claire. 16
2.7 Initial decision results in Table tennis, where the white, black and gray region express the uniform, non-uniform and undecided region. 16
2.8 Undecision rate for each refinement stage in Claire, where $N_0 = N_1 = 2$. 17
2.9 Block diagrams of GMM-based classifiers. 19
2.10 Block diagrams of Initial decision and refinement process. 20
2.11 Relation between the number of LUTs/FFs and the number of refinement. 22

3.1 An example of intervals with undesirable GMMs. 24
3.2 Relation between the true value (left-hand side) and round-down value (right-hand side). Round-down value becomes 0.5. 26
3.3 Relation between the true value (left-hand side) and round-up value (right-hand side). Round-up value becomes 0.75. 26
3.4 Original interval calculation classifier. 28
3.5 Proposed interval calculation classifier. 28

4.1 Simulation results of MAE obtained with $\mu_1 = 5.0$ and $\Sigma_1 = 0.2$. . . 37
4.2 Simulation results of MAE obtained with $\mu_1 = 1.0$ and $\Sigma_1 = 2.0$. . . 38
4.3 The results of computational time. 39

List of Tables

2.1 Values of $T[i]$ and $T[i]^{-1}$. 12
2.2 Average computational time per feature vector for evaluating Eq.(2.4) on DM6437 EVM board. 18
2.3 Computational time estimation by place-and-route timing report. . . 21
2.4 Area estimation by map report. 21

3.1 Results of logic synthesis for 32-bit implementation. 29
3.2 Results of logic synthesis for 64-bit implementation. 29
3.3 Results of throughput and clock latency for 32-bit implementation. . 29
3.4 Results of throughput and clock latency for 64-bit implementation. . 29

4.1 Values of the LUT. 34
4.2 Values of the scaled LUT. 35
4.3 Parameters of a mixture Gaussian distribution for generating normal random numbers. 36
4.4 Initial values used in the EM algorithm. 36

Bibliography

[1] C. Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, Vol. 2, pp. 246–252, 1999.

[2] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, Vol. 2, pp. 28 – 31, 2004.

[3] S.L. Phung, A. Bouzerdoum, and D. Chai. Skin segmentation using color pixel classification: analysis and comparison. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 27, No. 1, pp. 148 –154, 2005.

[4] R. Yagi, T. Kajimoto, and T. Nishitani. GMM foreground segmentation processor based on address free pixel streams. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pp. 1653–1656, 2012.

[5] S.J.D. Prince. *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012.

[6] Yu-Min Zeng, Zhen yang Wu, T. Falk, and W.-Y. Chan. Robust GMM based gender classification using pitch and RASTA-PLP parameters of speech. In *Machine Learning and Cybernetics, 2006 International Conference on*, pp. 3376 –3379, 2006.

[7] D.A. Reynolds and R.C. Rose. Robust text-independent speaker identification using gaussian mixture speaker models. *Speech and Audio Processing, IEEE Transactions on*, Vol. 3, No. 1, pp. 72 –83, 1995.

[8] W.M. Campbell, D.E. Sturim, and D.A. Reynolds. Support vector machines using gmm supervectors for speaker verification. *Signal Processing Letters, IEEE*, Vol. 13, No. 5, pp. 308 – 311, 2006.

[9] M. Fujimoto and Y.A. Riki. Robust speech recognition in additive and channel noise environments using gmm and em algorithm. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, Vol. 1, pp. I – 941–4 vol.1, 2004.

[10] Minghua Shi, A. Bermak, S. Chandrasekaran, and A. Amira. An efficient FPGA implementation of gaussian mixture models-based classifier using distributed arithmetic. In *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, pp. 1276 –1279, 2006.

[11] Dongbing Gu. Distributed EM algorithm for gaussian mixtures in sensor networks. *Neural Networks, IEEE Transactions on*, Vol. 19, No. 7, pp. 1154–1166, 2008.

[12] P. Gupta and P.R. Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, Vol. 46, No. 2, pp. 388–404, 2000.

[13] A.K. Jain, R.P.W. Duin, and Jianchang Mao. Statistical pattern recognition: a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 22, No. 1, pp. 4 –37, 2000.

[14] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing, 3rd Ed.* Cambridge Univ. Pr., 2007.

[16] Minghua Shi and A. Bermak. An efficient digital vlsi implementation of gaussian mixture models-based classifier. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 14, No. 9, pp. 962–974, Sept.

[17] A. P. Dempster, N. M Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of The Royal Statistical Society, Series B*, Vol. 39, pp. 1–22.

[18] Simone Fiori. Hybrid independent component analysis by adaptive lut activation function neurons. *Neural Networks*, Vol. 15, No. 1, pp. 85 – 94, 2002.

[19] S. Fiori. Generation of pseudorandom numbers with arbitrary distribution by learnable look-up-table-type neural networks. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pp. 1787 –1792, June 2008.

[20] Simone Fiori. Fast statistical regression in presence of a dominant independent variable. *Neural Computing and Applications*, pp. 1–12, 2012. In press.

[21] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *Trans. on Modeling and Computer Simulation*, Vol. 8, No. 1, pp. 3–30, 1998.

[22] G. E. P. Box and Mervin E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, Vol. 29, No. 2, pp. 610–611, 1958.

[23] Uwe Mayer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays, 3rd Ed.* Springer, 2010.

[24] Mitsuru Takahashi, Shogo Muramatsu, and Hisakazu Kikuchi. Area reduction of quadratic form computation by rag algorithm for gaussian-mixture-model classifiers. pp. 79–80, 2010. In Japanese.

Biography

Hidenori Watanabe received B.E. and M.E. degrees in electrical engineering from Niigata University in 2008 and 2010, respectively. He is currently a Ph.D. candidate at Niigata University. His research interests are in digital signal processing. Mr. Watanabe is a member of the Institute of Information, Electronics and Communication Engineers (IEICE) of Japan and a member of the Institute of Electrical and Electronics Engineers, Inc. (IEEE) of USA.

Research Works

Academic Papers

1. Hidenori Watanabe and Shogo Muramatsu, **"Fast Algorithm and Efficient Implementation of GMM-Based Pattern Classifiers,"** *Journal of Signal Processing Systems,* Vol 63, Issue 1, pp. 107-116, Apr. 2011.

2. Hidenori Watanabe and Shogo Muramatsu, **"Look-Up-Table-Based Exponential Computation and Application to an EM Algorithm for GMM,"** *IEICE Transactions on Fundamentals,* accepted.

International Conferences

1. Shogo Muramatsu and Hidenori Watanabe, **"Fast Algorithm for GMM-Based Pattern Classifier,"** *Proc. of 2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2009),* pp.633-636, Apr. 2009.

2. Hidenori Watanabe, Shogo Muramatsu and Hisakazu Kikuchi, **"MULTIPLIERLESS REFINEMENT SCHEME FOR INTERVAL CALCULATION OF GMM-BASED CLASSIFICATION,"** *Proc. on 2009 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC 2009),* pp. 282-285, Oct. 2009.

3. Hidenori Watanabe, Shogo Muramatsu and Hisakazu Kikuchi, **"Interval Calculation of Em Algorithm for GMM Parameter Estimation,"** *Proc. on 2010 IEEE International Symposium on Circuits and Systems (ISCAS 2010),* pp. 2686-2689, May. 2010.

4. Hidenori Watanabe, Yuji Kikuchi, Shogo Muramatsu, Toshiro Oitate, Mitsuyoshi Murata and Takamasa Suzuki, **"Statistical Edge Detector with GMM Classifier,"** *26th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC2011),* pp. 611-614, Jun. 2011.

Domestic Conferences

1. Shogo Muramatsu, Tsutomu, Watanabe and Hidenori Watanabe, **"Efficient Classification for Gaussian Mixture Models with Interval Calculation,"** *IEICE Technical Report 107(475),* pp. 55-60, Jan. 2008. (In Japanese.)

2. Ryunosuke Takeda, Shogo Muramatsu, Hidenori Watanabe and Hisakazu Kikuchi, **"An Efficient Classification Module Based on Gaussian Mixture Models with Interval Calculation,"** *IEICE General Conference 2008,* p. 110, Mar. 2008. (In Japanese.)

3. Ryunosuke Takeda, Shogo Muramatsu, Yuichiro Takahashi, Hidenori Watanabe and Hisakazu Kikuchi, **"An LSI Architecture of Classification Module for Gaussian Mixture Models with Interval Computations,"** *The 21st Workshop on Circuits and Systems in Karuizawa,* pp. 167-170, Apr. 2008. (In Japanese.)

4. Hidenori Watanabe, Shogo Muramatsu and Hisakazu Kikuchi, **"Implementation of Classification Circuit for Gaussian Mixture Models with Interval Calculation,"** *Proceedings of 10th DSPS Educators Conference,* pp. 80-81, Sep. 2008. (In Japanese.)

5. Takahiro Sato, Shogo Muramatsu, Hidenori Watanabe and Hisakazu Kikuchi, **"Interval Calculation with Fixed Point Arithmetic for Classification of Gaussian Mixture Models,"** *IEICE Society Conference 2008,* p. 63, Sep. 2008. (In Japanese.)

6. Hidenori Watanabe, Shogo Muramatsu and Hisakazu Kikuchi, **"Improvement ofDynamic Range for Interval Calculation Classifier of Gaussian Mixture Models,"** *2008 Convention Record, The Shin-Etsu Chapter of The IEICE,* p. 28, Sep. 2008. (In Japanese.)

7. Hidenori Watanabe and Shogo Muramatsu, **"DSP/FPGA Implementation of Gaussian-Mixture-Model-based Classifier with Interval Calculation,"** *The 22nd Workshop on Circuits and Systems in Karuizawa,* pp. 392-397, Apr. 2009. (In Japanese.)

8. Takahiro Sato, Shogo Muramatsu, Hidenori Watanabe and Hisakazu Kikuchi, **"Performance Evaluation of Interval Calculation with Fixed Point Arithmetic,"** *IEICE Society Conference 2009,* p. 71, Sep. 2009. (In Japanese.)

9. Hidenori Watanabe, Shogo Muramatsu and Hisakazu Kikuchi, **"GMM-based Classification with Interval Calculation using Multiplierless Refinemt Scheme,"** *IEICE Society Conference 2009,* p. 72, Sep. 2009. (In Japanese.)

10. Hidenori Watanabe, Shogo Muramatsu and Hisakazu Kikuchi, **"EM Algorithm Based on Interval Calculation for GMM,"** *24th IEICE Signal Processing Symp.,* pp. 203-208, Nov. 2009. (In Japanese.)

11. Hidenori Watanabe, Shogo Muramatsu and Hisakazu Kikuchi **"Fast HOG with Approximation of Gradient Intensity and Square of Histogram,"** *The 23rd Workshop on Circuits and Systems in Karuizawa,* pp. 131-136, Apr. 2010. (In Japanese.)

12. Hidenori Watanabe, Shogo Muramatsu and Hisakazu Kikuchi **"Adaboost with Interval Calculation based GMM Classifier,"** *25th IEICE Signal Processing Symp.,* pp. 459-462, Nov. 2010. (In Japanese.)

13. Hidenori Watanabe, Shogo Muramatsu, Hisakazu Kikuchi and Shigenobu Sasaki, **"Error Correcting Output Code through GMM-based Classifier with Interval Calculation,"** *26th IEICE Signal Processing Symp.,* pp. 545-549, Nov. 2011. (In Japanese.)

Patents

1. Shogo Muramatsu and Hidenori Watanabe, **"Identification Device, Identification Method, and Identification Processing Program,"** *Japan Patent No. 5112454 (U.S. Patent No. 8321368),* 27 Nov. 2012.