Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 917354, 22 pages
doi:10.1155/2009/917354

Research Article

A Reconfigurable Architecture for Rotation Invariant Multi-View Face Detection Based on a Novel Two-Stage Boosting Method

Jinbo Xu,[1] Yong Dou,[1] and Zhengbin Pang[2]

[1] *National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha 410073, China*
[2] *Institute of Computer, School of Computer, National University of Defense Technology, Changsha 410073, China*

Correspondence should be addressed to Jinbo Xu, xjb.nudt@gmail.com

Received 30 December 2008; Revised 9 May 2009; Accepted 19 August 2009

Recommended by Liang-Gee Chen

We present a reconfigurable architecture model for rotation invariant multi-view face detection based on a novel two-stage boosting method. A tree-structured detector hierarchy is designed to organize multiple detector nodes identifying pose ranges of faces. We propose a boosting algorithm for training the detector nodes. The strong classifier in each detector node is composed of multiple novelly designed two-stage weak classifiers. With a shared output space of multicomponents vector, each detector node deals with the multidimensional binary classification problems. The design of the hardware architecture which fully exploits the spatial and temporal parallelism is introduced in detail. We also study the reconfiguration of the architecture for finding an appropriate tradeoff among the hardware implementation cost, the detection accuracy, and speed. Experiments on FPGA show that high accuracy and marvelous speed are achieved compared with previous related works. The execution time speedups range from 14.68 to 20.86 for images with size of 160×120 up to 800×600 when our FPGA design (98 MHz) is compared with software solution on PC (Pentium 4 2.8 GHz).

1. Introduction

Over the years, great advances have been achieved on face detection research [1], which has already been widely applied in many real-world applications, such as biometrics, visual surveillance, human-computer interaction, to name a few. Some early works on face detection, for instance, Rowley's ANN method [2] and Schneiderman's method based on Bayesian decision rule [3], have achieved high accuracy, but their applications are very limited due to the tremendous computational load. The break through happened in 2001 when Viola and Jones [4] developed their Boosted Cascade Framework whose remarkable performance owes to the fast speed of Haar-like feature calculation based on the integral image, the high accuracy of boosted strong classifiers, and the asymmetric decision making of the cascade structure. Frontal face images at 384×288 resolutions were reported to be fairly reliably detected at 15 frames per second by using PC with PIII 700 MHz CPU.

Although many early researches have good performance in detection of frontal faces, multiv-iew face detection (MVFD) remains a challenging problem that needs more attention due to the much more complicated variation within the multi-view face class. MVFD is used to detect upright faces in images that with $\pm 90^\circ$ rotation-out-of-plane (ROP) pose changes. Rotation Invariant MVFD (RIMVFD) means to detect faces with both $\pm 90^\circ$ ROP and 360° rotation-in-plane (RIP) pose changes. Statistics show that most of the faces in images and videos of real world are nonfrontal [5], and therefore, the ability to deal with multi-view faces is important for many face-related applications.

In the past few years, many derivatives of Viola's work have been proposed for rotation invariant frontal face detection and MVFD. These derivatives can be categorized into four aspects: the detector structure, designing of strong classifiers, training of weak classifiers, and selecting of features. For the detector structure, Viola's cascade detector structure is extended by Wu's parallel cascades structure [6], Li's pyramid structure [7], Jones's decision tree [8], and Huang's WFS (width-first-search) tree in [9]. For designing strong classifiers, the discrete AdaBoost was replaced by real AdaBoost [10], Gentle Boost [11], boosting chain learning [12], FloatBoost [13], and Vector Boosting [9]. On the level of weak classifiers, the simple threshold-type function is

replaced by finer partition of the feature space such as piecewise functions [6] or joint binarizations of Haar-like features [14]. As for the feature level in particular, there are works of Lienhart's extended Haar-like feature set [15], Liu's Kullback-Leibler Boosting [16], Baluja's pair-wise points [17], Wang's RNDA algorithm [18], and Abramson's control point [19]. Most of these works focus on increasing the detection accuracy. However, these methods are only evaluated by using software solution, which can hardly achieve real-time MVFD for some time-constraint applications. There is still much work to do from the algorithm design to the ultimate practical systems.

In order to meet the needs of various applications, using dedicated hardware to accelerate RIMVFD is an effective solution. Without losing detection accuracy, there are several valuable advantages with dedicated hardware solution: (1) compared with the software solution, significant speedups of execution time could be achieved by fully exploiting temporal and spatial parallelism; (2) the system can be dynamically reconfigured to meet different requirements on accuracy, speed, and resources for various applications; (3) low power consumption and high mobility can be achieved, which is very useful for small battery driven devices and handheld devices; (4) the cost could be low for commercial perspective. In literature, there are hardware implementations of frontal face detection based on tone color detection [20, 21], Neural Networks [22], and AdaBoost [23]. However, few researches focus on hardware implementation of RIMVFD. Since different applications have different demands on the accuracy, speed, and resources cost, it is necessary to research on the reconfigurable hardware architecture for RIMVFD.

In this paper, a fine-classified method and an FPGA-based reconfigurable architecture for RIMVFD are presented. Firstly, a tree-structured detector hierarchy for RIMVFD is designed to organize detector nodes for both RIP and ROP pose changes. To train branching nodes of the detector tree, a fine-classified boosting algorithm with a novel two-stage weak classifier design is proposed. Then, the temporal and spatial parallelism of the proposed RIMVFD method is fully exploited. Next, the reconfigurable architecture for RIMVFD is designed and implemented on FPGA. The correlation among the hardware implementation cost, the detection accuracy and speed is evaluated, so that the system can be correctly reconfigured for different applications with different requirements. The main contributions of this paper are

(i) RIMVFD with all $\pm 90^\circ$ ROP and 360° RIP pose changes is achieved by using tree-structured detector hierarchy and fine-classified boosting method;

(ii) the execution time of the classification procedure is significantly reduced by fully exploiting the parallelism;

(iii) by dynamically reconfiguring the hardware architecture, the tradeoff among the hardware implementation cost, the detection accuracy, and speed is well tuned, so that the proposed design can easily meet the demands of different applications.

The remainder of this paper is organized as follows. In Section 2, the proposed RIMVFD method is described. Section 3 presents the hardware architecture model for the proposed RIMVFD on FPGA. The reconfigurable characteristics of architecture are analyzed in Section 4. Section 5 gives the experimental results. Concluding remarks are presented in Section 6.

2. Proposed RIMVFD Method

2.1. Review of AdaBoost. The basic idea introduced by Schapire and Freund [24–26] is that a combination of single rules or "weak classifiers" gives a "strong classifier." A training procedure uses a training sample set to obtain multiple weak classifiers iteratively.

We define the weighted learning set $\mathbf{S}$ of p samples as

$$\mathbf{S} = \left\{(\mathbf{x}_1, y_1, w_1), (\mathbf{x}_2, y_2, w_2), \ldots, \left(\mathbf{x}_p, y_p, w_p\right)\right\}. \quad (1)$$

The ith sample is defined by a feature vector $\mathbf{x}_i = (c_1, c_2, \ldots, c_D)^T$ in a D-dimensional space, its corresponding class $C(\mathbf{x}_i) = y_i \in \{-1, +1\}$ in the binary case, and the weight of the sample w_i, where c_j in $\mathbf{x}_i$, $i = 1, 2, \ldots, p$, and $j = 1, 2, \ldots, D$, represents the ith sample's feature value at the jth dimension.

A weak classifier h_t obtains a hypothesis whether a sample belongs to a class, $h_t(\mathbf{x})$, according to the feature values in the D-dimensional space. For example, whether a sample belongs to a subwindow in an image can be hypothesized by using the horizontal and vertical coordinates $(c_1, c_2)^T$ as feature values in a 2-dimensional space. If $y_i \neq h_t(\mathbf{x}_i)$, the weak classifier h_t makes a wrong decision on the ith sample; otherwise, the decision is right.

Each iteration of the process consists in finding the best possible weak classifier, that is, the classifier for which the error is minimum. After each iteration, the weights of the misclassified samples are increased, and the weights of the well-classified sample are decreased. The iterative procedure stops if $\varepsilon_t = 0$ or $\varepsilon_t \geq 1/2$. The reason is in [25]: in the case of binary classification ($k = 2$), a weak hypothesis h_t with error significantly larger than 1/2 is of equal value to one with error significantly less than 1/2 since h_t can be replaced by 1–h_t; and for $k > 2$, a hypothesis h_t with error $\varepsilon_t \geq 1/2$ is useless to the boosting algorithm. If such a weak hypothesis is returned by the weak learner, the algorithm simply halts, using only the weak hypotheses that were already computed (i.e., set the number of weak classifiers in the strong classifier $T = t$–1).

The final strong classifier f is given by

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{t=1}^{T} a_t h_t(\mathbf{x})\right), \quad (2)$$

where both α_t and h_t are to be learned by the boosting procedure presented in Algorithm 1.

2.2. AdaBoost-Based Face Detection Framework. Accepted by the computer vision community as the state of the art in terms of speed and accuracy, face detection based on

(1) Input $\mathbf{S} = \{(\mathbf{x}_1, y_1, w_1), (\mathbf{x}_2, y_2, w_2), \ldots, (\mathbf{x}_p, y_p, w_p)\}$, number of iterations T.
(2) Initialize $w_i^{(0)} = 1/p$ for all $i = 1, \ldots, p$.
(3) Do for $t = 1, \ldots, T$
  (3.1) Train classifier with respect to the weighted samples set and obtain hypothesis
$$h_t : \mathbf{S} \rightarrow \{-1, +1\}.$$
  (3.2) Calculate the weighted error ε_t of h_t:
$$\varepsilon_t \sum_{i=1}^{p} w_i^{(t)} I(y_i \neq H_t(\mathbf{x}_i)).$$
  (3.3) Compute the coefficient α_t:
$$\alpha_t = \frac{1}{2} \log\left(\frac{(1-\varepsilon_t)}{\varepsilon_t}\right).$$
  (3.4) Update the weights
$$w_i^{(t+1)} = \frac{w_i^{(t)}}{Z_t} \exp\{-\alpha_t y_i h_t(\mathbf{x}_i)\},$$
  where z_t is a normalization constant:
$$Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$$
(4) Stop if $\varepsilon_t = 0$ or $\varepsilon_t \geq 1/2$. Set $T = t - 1$.
(5) Output: $f(x) = \operatorname{sgn}\left(\sum_{t=1}^{T} a_t H_t(x)\right)$.

Algorithm 1: The basic AdaBoost procedure.

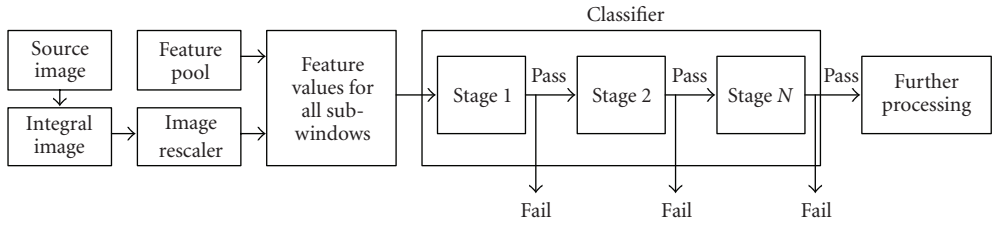

Figure 1: Framework of AdaBoost-based face detection.

AdaBoost algorithm presents two significant advantages; it has the ability to quickly eliminate nonface regions, and the classification process itself is extremely fast.

While Haar wavelets were used in [27] for representing faces and pedestrians, Viola and Jones proposed the use of Haar-like features which can be computed efficiently with integral image [28]. Each feature consists of a set of black and white rectangles. The result of a feature is the sum of the pixels under the white rectangle minus the sum of the pixels under the black rectangle. In Viola's cascade framework, a group of features composes a classification stage based on AdaBoost. The outcome of a stage determines whether the region of the image examined contains a face or not. The size of the feature determines the size of the image region being searched. When the base size is processed for all regions, the features are then enlarged in subsequent scales, and evaluated for each scale to be able to detect faces of larger sizes. Additional algorithm details can be found in [28]. The algorithm outline is shown in Figure 1.

The difference between our RIMVFD method and the framework in Figure 1 is mainly reflected in the classifier part. The integral image computation and image rescaler are the same, which are widely used in face detection domain currently. In this paper, we will focus on the classifying strategy of RIMVFD.

2.3. Tree-Structured Detector Hierarchy for RIMVFD. For RIMVFD, new detector hierarchy different from the cascaded structure for frontal face detection has to be designed, since pose changes should be identified in addition to the face/nonface classification. Huang et al. [9] reviewed some traditional structures for organizing detector nodes, such as parallel cascades structure [6], pyramid structure [7], and decision tree method [8]. After having analyzed the weaknesses of these structures, Huang proposed a WFS-tree structure, who claimed that the structure can balance the face/nonface distinguishing and pose identification tasks, so as to enhance the detector in both accuracy and speed. Because of limited parallel processing ability in general-purpose processor, Huang's tree structure only divides all faces into 5 categories according to ROP to control the computational cost, which is not so fine-grained.

In this paper, we extend Huang's tree structure so as to achieve more accuracy and speed with the aid of reconfigurable hardware acceleration. The extended tree structure covers all $\pm 90^\circ$ ROP and 360° RIP pose changes. The coarse-to-fine strategy is adopted to divide the entire face space into smaller and smaller subspaces as shown in Figure 2.

Whether a face sample belongs to a pose range or not is determined by the output of a detector node corresponding to the current pose range. The output of the detector node is a Boolean value $g(x)$. In Figure 2, all detector nodes which have the same parent node are trained with the same training sample set belonging to the parent pose range. The outputs of these detector nodes form a determinative vector $\mathbf{G}(x) = (g_1(x), \ldots, g_k(x))$, where k is the number of the detector nodes with the same parent node. When the sample reaches

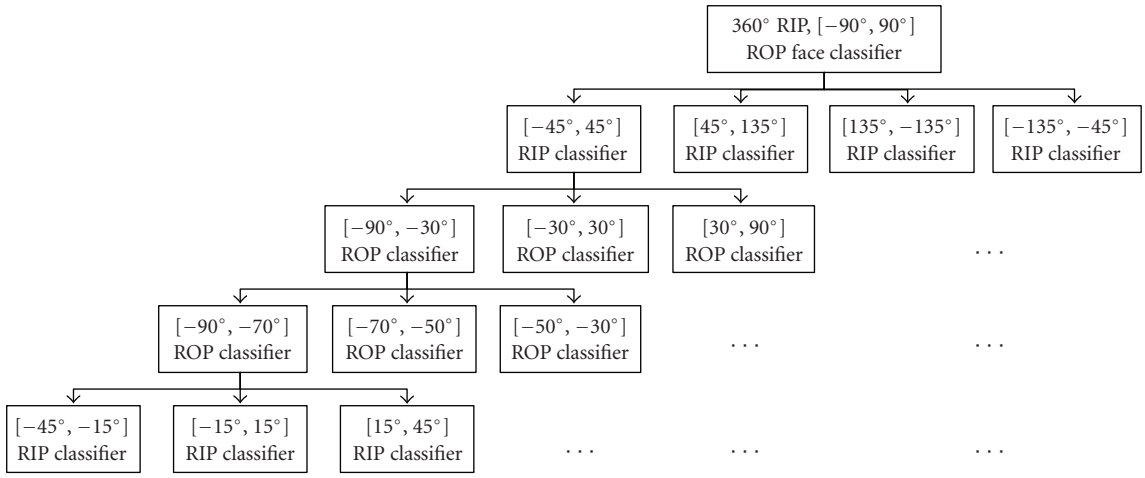

Figure 2: Illustration of the proposed fine-grained tree structure.

the leaf nodes, the RIP and ROP ranges that the sample lies in are determined, from which the pose of the sample is classified.

The main extensions of the tree structure different from the one in [9] are that the ROP and RIP granularity are refined for more accuracy. And the number of detector nodes as shown in Figure 2 can be reduced about 75% (i.e., from 161 to 41 for this contribution) by using the reconfigurable computing techniques, which will be further described in Section 3.1. With the aid of hardware acceleration, the detection speed is guaranteed.

2.4. Fine-Classified Boosting Method for RIMVFD. After designing the detector hierarchy, we need to research on the classification method for each detector node. Intrinsically, the determination of vector $\mathbf{G}(x)$ is the key problem. Take the $(-45^\circ, 45^\circ)$, RIP detector node in Figure 2 as an example. A sample which passes through this node will be sent into $(-90^\circ, -30^\circ)$, ROP, $(-30^\circ, 30^\circ)$, ROP and $(30^\circ, 90^\circ)$, ROP detector nodes in the subsequent level. At this moment, $\mathbf{G}(x)$ is a 3-dimensional vector. For example, $\mathbf{G}(x) = (1, 0, 1)$ means that the sample may lie in either $(-90^\circ, -30^\circ)$, ROP or $(30^\circ, 90^\circ)$, ROP pose range. This section proposes a fine-classified boosting method to get $\mathbf{G}(x)$ fast and accurately. The designing and training of the proposed method are detailed in this section.

Many derivations from the basic AdaBoost method have been proposed to be used for MVFD. Schapire and Singer presented a multiclass multilabel (MCML) version of the Real AdaBoost [10], which assigns a set of labels for each sample and decomposes the original problem into k orthogonal binary ones. Huang et al. proposed the Vector Boosting [9] in which both its weak learner and its final output are vectors rather than scalars. In Huang's method, Vector Boosting will deal with the decomposed binary classification problems in a unified framework by means of a shared output space of multicomponents vector. Each binary problem has its own "interested" direction in this output space that is denoted as its projection vector. For each binary problem, a confidence value is calculated by using an extended version of the Real AdaBoost. Then the confidence vector composed of these confidence values is compared with a threshold vector $\mathbf{B}$ to obtain a Boolean output vector, where each vector element indicates whether a sample belongs to the corresponding class. This kind of method only considers the confidence value for the current class while determining whether a sample belongs to this class, and does not consider confidence values for the other classes. The disadvantage is that the number of the candidate classes that a sample may belong is poorly constrained into an ideal extent, since a sample may be classified into one class by using the threshold in this dimension, and then classified into another class by using the threshold in that dimension. Actually, a face sample should only belong to a single pose class for face detection applications. Take the previous example in the first paragraph, for a sample which lies in $(-90^\circ, -30^\circ)$ ROP pose range, $\mathbf{G}(x)$ is expected to be (1, 0, 0). However, it may be wrongly calculated as (1, 0, 1), which decreases the detection efficiency obviously.

The fine-classified boosting method proposed in this paper has the advantage of decreasing misclassified cases. For example, $\mathbf{G}(x) = (1, 0, 1)$ mentioned before may be refined to be (1, 0, 0) by using the proposed method. The novelty of the method is that each weak classifier in the detector tree is trained by using a two-stage boosting mechanism. The training of all k detector nodes with the same parent node uses the same training sample set. We use $\mathbf{h}_t = \{h_{t(i)} \mid i = 1, \ldots, k\}$ to denote the vector composed of the tth weak classifiers in all these k detector nodes. The first stage of $\mathbf{h}_t(x) : \mathbf{h}_t' : \chi \to \mathbb{R}^k$ is based on piece-wise decision function, where x belongs to a sample space χ, and $\mathbb{R}^k$ is k-dimensional real-value confidence space. The decision function in this stage is used to obtain a k-dimensional real-value confidence vector $\{c_i \mid i = 1, \ldots, k\}$ for each sample according to its Haar-like feature. Each confidence value indicates the probability that the sample belongs to the corresponding pose class (e.g., the first, second, and third confidence value can be used to correspond to the left,

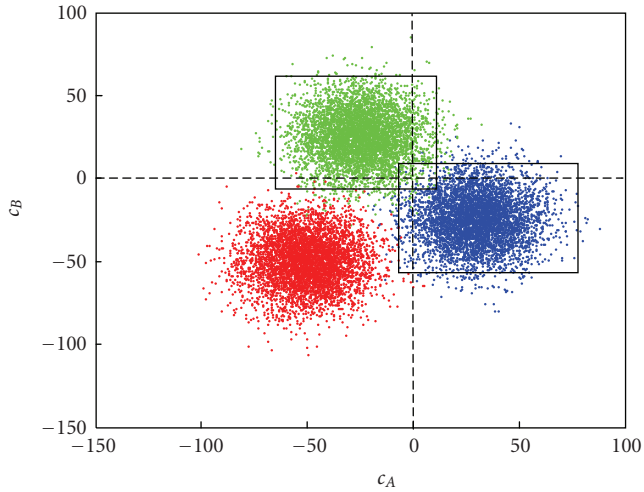

Figure 3: Distribution of the samples in the 2D output space.

frontal and right face class). One Haar-like feature is used for all k detector nodes with the same parent during each iteration of training procedure. The k-dimensional real-value confidence vector $\{c_i \mid i = 1, \ldots, k\}$ is then used as the input feature vector for the second stage of $\mathbf{h}_t(x)$. The second stage $\mathbf{h}_t'' : \mathbb{R}_k \rightarrow \{-1 \mid +1\}^k$ is based on hyper-rectangle decision function. It is trained to more precisely discriminate different dimensions by using the real confidence values generated in the first stage. The output is a k-dimensional Boolean vector, where the ith element gives the determination whether a sample belongs to the ith pose class. In this stage, each output in the k-dimensional Boolean vector depends on all k elements in the real-value confidence vector. In other words, while deciding whether a sample belongs to the ith face class, not only the probability that the sample belongs to the ith face class but also the probability that the sample does not belong to the other face classes should be considered. Since the full-mesh mapping strategy between the two stages considers the correlation of different confidence values in different dimensions, the pose class where a sample lies can be classified more precisely. However, $\mathbf{h}_t' : \boldsymbol{\chi} \rightarrow \mathbb{R}^k$ and $\mathbf{h}_t'' : \mathbb{R}_k \rightarrow \{-1 \mid +1\}^k$ are combined to form the final weak classifier $\mathbf{h}_t(x) : \boldsymbol{\chi} \rightarrow \{-1 \mid +1\}^k$ in the current iteration, which estimates whether a sample belongs to the ith class or not, where $i = 1, \ldots, k$.

Take a 2-dimensional classification problem as an example. The samples in the training set either belong to face class A or face class B or nonface. Then each sample will be assigned a 2D confidence vector (c_A, c_B), where c_A indicates the probability that the sample belongs to face class A, and c_B is similar. The distribution of the samples can be illustrated as shown in Figure 3, where (c_A, c_B) is treated as the 2D coordination of each sample. The blue dots are samples in face class A, and the green ones are samples in face class B, and the red ones are nonfaces. The vertical and horizontal dashed lines in Figure 3 illustrate the classification criterion based on single threshold, while the two rectangles illustrate the hyper-rectangle-based classification criterion. Intuitively, the latter generates less misclassified samples than the former. For example, for the green dots which lie in the left rectangle and below the horizontal dashed line, they are supposed to be classified as class B faces. However, they are considered as nonfaces by the single-threshold classifiers, which is misclassified. On the contrary, the hyper-rectangle-based classifiers can give the right decisions.

The weak learner is called repeatedly under the updated distribution to obtain multiple weak classifiers, which finally form a highly accurate classifier. A k-dimensional shared output space $\mathbf{H}(x)$ is the linear combination of all trained weak classifiers. An $n \times k$ matrix $\mathbf{A}$ made up of n k-dimensional projection vectors is used to transform the k-dimensional output space into n-dimensional confidence space $\mathbf{F}(x)$. Each dimension of the confidence space corresponds to a certain pose class. Finally, the output of the strong classifier is the sign of $\mathbf{F}(x)$.

Take the previous example in the first paragraph, $(-90^\circ, -30^\circ)$ ROP, $(-30^\circ, 30^\circ)$ ROP, and $(30^\circ, 90^\circ$ ROP detector nodes use the same training sample set to get the classifiers, where the dimension of the shared output space, k, is 3. If we are dealing with the $(-90^\circ, -30^\circ)$ ROP and $(30^\circ, 90^\circ)$ ROP classification problems, n is 2, and $n \times k = 2 \times 3$ matrix $\mathbf{A}$ is ((1,0,0),(0,0,1)) consequently. The projection vector (1,0,0) extracts 1st element from $\mathbf{H}(x)$ for $(-90^\circ, -30^\circ)$ ROP pose classification. Similarly, the projection vector (0,0,1) extracts third element from $\mathbf{H}(x)$ for $(30^\circ, 90^\circ)$ ROP pose classification.

Algorithm 2 gives the generalized description of the fine-classified boosting method. We define the sequence of training set of m samples as $\mathbf{S} = \{(x_1, \boldsymbol{\nu}_1, y_1), \ldots, (x_m, \boldsymbol{\nu}_m, y_m)\}$, where x_i belongs to a sample space $\boldsymbol{\chi}$, $\boldsymbol{\nu}_i$ belongs to a finite k-dimensional projection vector set $\boldsymbol{\Omega}$ which indicates a certain pose class, and the label $y_i = \pm 1$ indicates whether the sample belongs to the pose class or not. For each sample, the margin of a sample x_i with its label y_i and projection vector $\boldsymbol{\nu}_i$ is defined as $y_i \cdot (\boldsymbol{\nu}_i \cdot \mathbf{h}_t(x_i))$. For example, $y_i = 1$ means that sample x_i belongs to $(-90^\circ, -30^\circ)$ ROP pose class which is defined by $\mathbf{v}_i = (1, 0, 0)$, while the classifier $\mathbf{h}_t(x_i)$ gives an opposite result: -1, which means that the classifier does not consider the sample belongs to $(-90^\circ, -30^\circ)$ ROP pose class. Therefore, the classifier makes a wrong estimation. At the moment, $y_i \cdot (\boldsymbol{\nu}_i \cdot \mathbf{h}_t(x_i)) = -1$. Otherwise, while the classifier makes a right decision, $y_i \cdot (\boldsymbol{\nu}_i \cdot \mathbf{h}_t(xi)) = 1$. $y_i \cdot (\boldsymbol{\nu}_i \cdot \mathbf{h}_t(x_i))$ is used to update the sample weights. With the introduction of $\mathbf{v}_i$, the orthogonal component of a weak classifier's output makes no contribution to the updating of the sample's weight. Consequently, it increases the weights of samples that have been wrongly classified according to the projection vector (in its "interested" direction) and decreases those with correct predictions.

Next, the training procedure of the two-stage weak classifier is described in detail.

2.5. Training of Weak Classifiers. As mentioned before, for each iteration of training weak classifiers, two stages are used. We present the training of these two-stage weak classifiers in this section.

For a classification problem that has n classes, given:
(1) Projection vector set $\boldsymbol{\Omega} = \{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n\}, \boldsymbol{\omega}_i \in \mathbb{R}^k$
(2) Sample set $\mathbf{S} = \{(x_1, \boldsymbol{\nu}, y_1), \ldots, (x_m, \boldsymbol{\nu}_m, y_m)\}$, where $x \in \boldsymbol{\chi}, \boldsymbol{\nu} \in \boldsymbol{\Omega}, y = \pm 1$
(I) Initialize the sample distribution $D_1(i) = 1/m$ for all $i = 1, \ldots, m$.
(II) For $t = 1, \ldots, T$
(i) Under current distribution, train the first stage of a weak classifier

$$\mathbf{h}_t^{'} : \boldsymbol{\chi} \rightarrow \mathbb{R}^k$$

(ii) Under current distribution, train the second stage of a weak classifier

$$\mathbf{h}_t^{''} : \mathbb{R}^k \rightarrow \{-1 \mid +1\}^k$$

(iii) Combine $\mathbf{h}_t^{'}$ and $\mathbf{h}_t^{''}$ with a full-mesh mapping strategy to form the weak classifier in the current iteration:

$$\mathbf{h}_t(x) : \boldsymbol{\chi} \rightarrow \{-1 \mid +1\}^k$$

(vi) Calculate the weighted error ε_t of $\mathbf{h}_t(x)$

$$\varepsilon_t = \sum_{i=1}^{m} D_t(i) \cdot I(y_i \neq \boldsymbol{\nu}_i \cdot \mathbf{h}_t(x_i))$$

(v) Compute the coefficient α_t:

$$\alpha_t = \frac{1}{2} \log\left(\frac{(1 - \varepsilon_t)}{\varepsilon_t}\right)$$

(vi) Update the sample distribution

$$D_{t+1}(i) = \frac{D_t(i) \cdot \exp(-\alpha_t \cdot y_i \cdot (\boldsymbol{\nu}_i \cdot \mathbf{h}_t(x_i)))}{Z_t},$$

where Z_t the normalization factor so as to keep D_{i+1} as a probability distribution.
(III) Stop if $\varepsilon_t = 0$ or $\varepsilon_t \geq 1/2$, Set $T = t$-1.
(IV) The final output space is

$$\mathbf{H}(x) = \sum_{t=1}^{T} \alpha_t \mathbf{h}_t(x).$$

(V) The confidence space is $\mathbf{F}(x) = \mathbf{A}\mathbf{H}(x)$, where the transformation matrix $\mathbf{A} = \{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n\}^T$.
(VI) The final strong classifier is $\mathbf{G}(x) = \operatorname{sgn}(\mathbf{F}(x))$.

Algorithm 2: A Generalized description of the fine-classified boosting method.

2.5.1. The First Stage of Weak Classifier. The first stage of weak classifier is used to get different real confidence values for k dimensions based on the sample's Haar-like feature. For the ith dimension, if a sample's Haar-like feature is in a range R_i, the sample is considered belonging to this class and a higher confidence value is assigned to the sample; otherwise, if a sample's feature is in another range $\overline{R_i}$, the sample is considered not belonging to this class and a lower confidence value is assigned. In this way, each sample is assigned a confidence vector $\{c_i \mid i = 1, \ldots, k\}$.

The classifier is based on piece-wise function, which is more efficient than threshold-type function and can be efficiently implemented with lookup table (LUT) [9]. It divides the feature space into many bins with equal width and outputs a constant real value for each bin.

The objective of weak learner is to minimize the normalization factor of current round if adopting greedy strategy. Following Viola's exhaustive search method in [4], we enumerate a finite redundant Haar-like feature set $\mathbf{P} = \{f_k(\mathbf{x}; \theta_k)\}$, $1 \leq k \leq n$, and optimize a piece-wise function for each feature so as to obtain the most discriminating one. Finally, for all detector nodes in the detector tree, 3820 Haar-like features are selected from about 48 700 features through the training procedure as shown in Algorithm 3. The problem is how to get the optimal piece-wise function μ. A piece-wise function is configured by two parts: one is the division of feature space, the other is the constant for each division (i.e., bin). For simplification, the first one is fixed for each feature empirically, and the output constant for each bin can be optimized as in Algorithm 3.

When a sample passes through the first stage $\mathbf{h}_t^{'} : \boldsymbol{\chi} \rightarrow \mathbb{R}^k$, a confidence vector is obtained. Each real-value element in the vector indicates the probability whether the sample belongs to the corresponding class. Then these confidence vectors for all training samples are sent into the second stage of weak classifier to finely discriminate different classes.

2.5.2. The Second Stage of Weak Classifier. The second stage $\mathbf{h}_t^{''} : \mathbb{R}^k \rightarrow \{-1| + 1\}^k$ is used to further determine whether a sample belongs to the ith class based on the real-value confidence vector indicating the probability that the sample belongs to each class. For a sample x, if its confidence vector is $\{c_i \mid i = 1, \ldots, k\}$, it means that the probability that the sample belongs to the ith class is c_i. Some related works consider the sample belonging to any class where the confidence value is above a single threshold. Usually, this causes the sample being classified into multiple classes, which will obviously decrease the accuracy and increase the computational cost. To narrow the candidate classes, our second stage of weak classifier will learn the distribution of these confidence vectors for all samples, and generate a precise criterion to discriminate different classes. The clear boundary among different classes can filter out most classes that the sample does not belong actually.

In this paper, the second stage is based on hyper-rectangles. It has been proved in the literature that decision trees based on hyper-rectangles (or union of boxes) instead of a single threshold give better results [29]. The decision function associated with a hyper-rectangle can be easily

Given: Sample set $\mathbf{S} = \{(x_1, \mathbf{v}_1, y_1), \ldots, (x_m, \mathbf{v}_m, y_m)\}$, finite feature set $\mathbf{P} = \{f_k(\mathbf{x}; \theta_k)\}$, $1 \le k \le n$, and the current distribution $D_t(i)$.

(I) for $k = 1, \ldots n$ (each feature)

(i) Group all samples into different bins

$$\mathbf{S}_j = \{(x_i, \boldsymbol{\nu}_i, y_i) \mid x_i \in \text{bin}_j\}, \text{bin}_j = [(j-1)/p, j/p], 1 \le j \le p$$

(ii) For $j = 1, \ldots, p$ (each bin)

Using Newton-step method to compute

$$\mathbf{c}_\mathbf{j}^* = \arg\min_{c_j}\left(\sum_{s_j} D_t(i) \exp(-y(\boldsymbol{\nu}_i \cdot \mathbf{c}_j))\right)$$

(iii) Create weak classifier $\mathbf{h}(x; \theta_k, \mu_k)$ based on $\{c_\mathbf{j}^*\}$

$$\mathbf{h}(x; \theta_k, \mu_k) = \sum_{\mathbf{j}=1}^{\mathbf{p}} \mathbf{c}_\mathbf{j}^* B_P^j(\theta_k),$$

$$\text{where } B_P^j(u) = \begin{cases} 1 & u \notin [((j-1)/p, j/p) \\ 0 & u \notin [((j-1)/p, j/p) \end{cases} \quad j = 1, \ldots, p.$$

and use equation

$$Z_t = \sum_j \sum_{S_j} D + t\ (i) \exp(-y(\boldsymbol{\nu}_i \cdot \mathbf{c}_j))$$

to calculate its corresponding normalization factor Z_t.

(II) Return the optimal weak classifier

$$\mathbf{h}_t'(x) = \arg\min_{\mathbf{h}_t(x)}(Z_t)$$

Algorithm 3: Training of the first stage of weak classifier with piece-wise function.

implemented in parallel. Each sample has a confidence vector $\{c_i \mid i = 1, \ldots, k\}$ in a k-dimensional space, and the goal is to output a Boolean-value for each dimension to determine whether the sample belongs to the corresponding class. We define the generalized hyper-rectangle as a set $\mathbf{H}$ of $2k$ thresholds and a class y_H, with $y_H \in \{-1, +1\}$:

$$\mathbf{H} = \left\{\theta_1^l, \theta_1^u, \theta_2^l, \theta_2^u, \ldots, \theta_k^l, \theta_k^u, y_H\right\}, \tag{3}$$

where θ_i^l and θ_i^u are the lower and upper limits of a given interval in the ith dimension. The decision function is

$$h_H(x) = y_H \Longleftrightarrow \prod_{i=1}^{k}\left(\left(c_i > \theta_i^l\right) \quad \text{and } \left(c_i < \theta_i^u\right)\right), \tag{4}$$

$$h_H(x) = -y_H, \text{otherwise.}$$

The core of the training of the second stage is the hyper-rectangle set $\mathbf{S}_H$ determination from a set of samples $\mathbf{S}$. We use the method proposed in [30] to train the weak classifier. The basic idea is to build around each sample $\{x_i, y_i\} \in \mathbf{S}$ a box or hyper-rectangle $\mathbf{H}(x_i)$ containing no sample of opposite classes, where x_i is the confidence vectors of all samples in this paper and $y_i = \pm 1$. Additional details of the training procedure can be found in [30].

3. Design of the Hardware Architecture Model

After having designed the detector hierarchy for RIMVFD, the strong classifier for each detector node and the weak classifier in each strong classifier, we will present the hardware architecture model for RIMVFD. The temporal and spatial parallelism of the proposed RIMVFD method is fully exploited on different design levels from the global structure to the weak classifiers.

3.1. Design of the Global Structure. As illustrated in Figure 1, the key parts of a face detection system based on Haar-like feature and AdaBoost are the classifier, the integral image computation and image rescaler. Although the work in[28] suggested that rescaling a frame is more expensive than rescaling the features, this is not always so. For an embedded system without a large cache, the loss of data locality incurred by the larger rescaled features is worse than the cost of including a small rescaling block, because loss of locality implies the main memory will be accessed for each use of integral image data, which is beyond the bandwidth affordable in a low-cost device. We compare the bandwidth requirements for main memory between the two rescaling methods here. For the method of re-scaling images, the sizes of feature subwindows for different rescaling levels are the same. While linearly transferring image data from main memory to onchip memory for caching, the cached data can be reused for adjacent subwindows if the subwindow is small enough. In this way, each pixel in the integral images in all rescaled levels is transferred only once, and the bandwidth requirement for processing a frame is proportional to the sum of size of integral images in all rescaled levels. Assuming that the scaling factor is r, the data size of the original integral image is C, and the number of re-scaled levels is l, the bandwidth requirements can be denoted as $BW_{\text{image}} = C(1 + (1/r^2) + (1/r^2)^2 + \cdots + (1/r^2)^{l-1})$. For the method of rescaling features, while the size of features is re-scaled large enough to a certain extent, the onchip memory may not simultaneously hold the data required for processing a single feature subwindow. Therefore, the system will have to access the main memory one more time before processing the next subwindow to transfer the uncached part of the current subwindow into the onchip memory. Consequently, the cached data in the onchip memory for processing the current subwindow cannot be reused for processing the next subwindow, which means that a lot of pixels in the

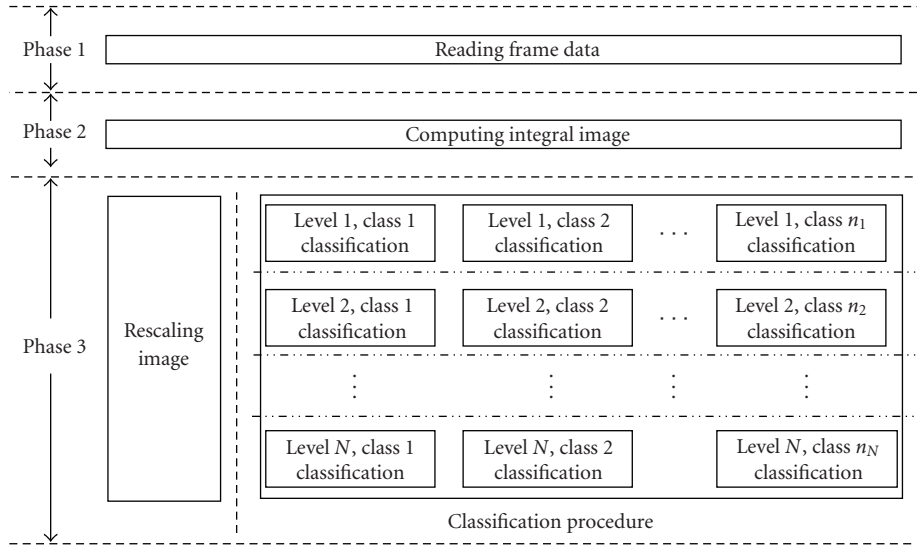

Figure 4: Skeleton of the system pipeline structure.

integral image will have to be transferred more than once from main memory to onchip memory. In this situation, the bandwidth requirements can be denoted as $BW_{\text{feature}} = C(a_1 + a_2 + \cdots + a_l)$, where $a_i \geq 1$, $i = 1, 2, \ldots, l$, and a_i means that some pixels in the integral image are transferred more than once. Obviously, $BW_{\text{feature}} > BW_{\text{image}}$, which means that for a system without large caches, the bandwidth requirements while rescaling features are larger than the bandwidth requirements while rescaling images. Therefore, in our work, we rescale the image instead of the features, which means that each iteration of rescaling has a corresponding rescaled integral image.

We consider the parallelism first. The phase of acquiring frames (*Phase 1*), computing integral frames (*Phase 2*), and detecting faces within frames (*Phase 3*) can form a pipeline, which works in a data-driven mechanism. A processing module starts operation once the needed data is available. The results will wake up the modules that are waiting for them. As shown in Figure 4, while *Phase 3* is processing the ith frame, *Phase 2* can start processing the $(i + 1)$th frame, and *Phase 1* can operate on the $(i + 2)$th frame simultaneously. At *Phase 3*, for a new frame, the input data for image rescaler and classification procedure are both generated from *Phase 2*, therefore, the image rescaler and the classification procedure can work in parallel, as illustrated in Figure 4. Each frame is required to be rescaled to multiple scales and be processed by the classification procedure at each scale. Therefore, at *Phase 3*, when the classification procedure is processing subwindows at the jth image scale, the image rescaler can generate rescaled integral image data at the $(j + 1)$th scale. After rescaled integral image at the $(j + 1)$th scale is generated, the data are sent to the classification procedure for processing. Simultaneously, the image rescaler starts generating the $(j+2)$th rescaled integral image. During the classification procedure at each scale, the feature calculation and different classification levels in the detector hierarchy work on different subwindows in a pipelined fashion, and classifiers for different classes in the same classification level work in parallel (as shown in the bottom right part of Figure 4). Figure 4 gives the skeleton of the system pipeline structure. Generally, for the tree-structured detector hierarchy in this paper, the numbers of classifiers in different levels are different. The lower the level is, the more the classifiers are. Consequently, $n_N > \cdots > n_2 > n_1$ in Figure 4.

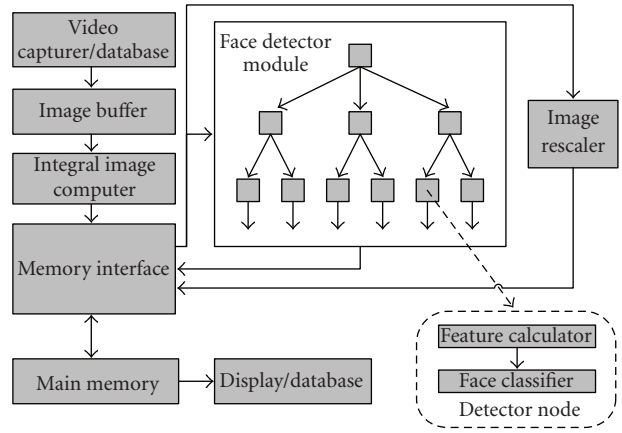

Figure 5: Global architecture of the RIMVFD system.

According to the parallel characteristics of the system, the global architecture for RIMVFD is designed as illustrated in Figure 5. The Image Buffer module buffers the image data acquired from the Video Capturer/Database. The Integral Image Computer processes the original image to get the integral image and stores the data into the Main Memory via the multiport Memory Interface. At the same time, the data are sent into the Face Detector Module and the Image Rescaler simultaneously. The Image Rescaler rescales the integral image into a demanded scale and writes the rescaled data back into the Main Memory for later use.

The Face Detector module uses the tree structure illustrated in Figure 2. In each node, a Feature Calculator and a Face Classifier are included. For a new image, the Face Detector module first retrieves integral image data from the Integral Image Computer module. The classification results which indicate the position, size, and pose of each detected faces are sent back into the Main Memory for display or other purposes. After the image in the original size is processed by the Face Detector module, the rescaled integral image data are read from the Main Memory and sent into the Face Detector module for detection of faces in the current scale. As long as there are rescaled integral image data in the Main Memory which are generated by the Image Rescaler module, the procedure continues until all scales are processed for this image. Then the system starts processing another image.

It can be seen that for an image, data that should be stored in the Main Memory include the integral image data from the Integral Image Computer module, the rescaled integral image data from the Image Rescaler module and the classification results from the Face Detector module. Also data that should be read from the Main Memory include the integral image data required by the Face Detector module and the classification results required by the Display/Database module. Assuming that the original image size is $x \times y$, and the depth of the integral image pixel is d Bytes, and the scaling factor is r $(r > 1)$, and the frame rate is f, and the classification results of an image occupies R Bytes, we can estimate the bandwidth requirements of the Main Memory as follows:

$$BW = 2 \times f \times \left(x \times y \times d \times \left(1 + \left(\frac{1}{r^2} \right)^2 + \cdots \right) + R \right) \text{Bytes}. \quad (5)$$

The experimental results on the bandwidth requirements are shown in Table 5.

For the computation of integral image and the rescaling of image, the hardware structures have no much differences with those in the frontal face detection system, which have been researched in many related works (e.g., [23, 31]). Therefore, we will not discuss them in this paper and focus on the design of the Face Detector Module.

The number of detector nodes as shown in Figure 2 can be reduced about 75% without significantly affecting the speed. Since any detector node in the $(45^\circ, 135^\circ)$, $(135^\circ, -135^\circ)$, and $(-135^\circ, -45^\circ)$ RIP ranges can be generated by rotating the corresponding detector node in the $(-45^\circ, 45^\circ)$ RIP range for 90°, 180°, and 270°, the detector trees for $(-45^\circ, 45^\circ)$, $(45^\circ, 135^\circ)$, $(135^\circ, -135^\circ)$, and $(-135^\circ, -45^\circ)$ RIP ranges can reuse the same structure. The only differences among them are the data acquisition addresses during the feature calculation procedure. Therefore, only the $(-45^\circ, 45^\circ)$ RIP classifier and its child nodes are configured into the FPGA chip at first, together with the root node (i.e., 360° RIP, $(-90^\circ, 90^\circ)$ ROP face classifier). If a subwindow is rejected by the root node, it is considered as non-face; otherwise, it flows into the $(-45^\circ, 45^\circ)$ RIP classifier. If it is approved, it is considered as an upright face ($(-45^\circ, 45^\circ)$ RIP) and will be further processed by the child nodes of $(-45^\circ, 45^\circ)$ RIP classifier until the final results are generated; otherwise, if it is rejected, the location of the subwindow is buffered for later processing. After all subwindows in the image are processed by the detector tree, the data acquisition scheme in the feature calculation part is updated to rotate the feature values for 90°, 180°, and 270°. In this way, the $(-45^\circ, 45^\circ)$ RIP detector tree turns into the detector trees for $(45^\circ, 135^\circ)$, $(135^\circ, -135^\circ)$, and $(-135^\circ, -45^\circ)$ RIP ranges, respectively. The buffered subwindows which are rejected by the $(-45^\circ, 45^\circ)$ RIP classifier are reprocessed by the new $(45^\circ, 135^\circ)$, $(135^\circ, -135^\circ)$, and $(-135^\circ, -45^\circ)$ RIP detector tree successively. Only the subwindows rejected by the 90° RIP classifier in the previous detector tree are sent into the successive one. Since most multi-view faces concentrate in $(-45^\circ, 45^\circ)$ RIP range in real world, the number of subwindows that are rejected by the $(-45^\circ, 45^\circ)$ RIP classifier and need to be reprocessed by the $(45^\circ, 135^\circ)$, $(135^\circ, -135^\circ)$, and $(-135^\circ, -45^\circ)$ RIP detector trees will be tiny. Consequently, although the number of detector nodes as shown in Figure 2 is reduced from 161 to 41, the detection speed is hardly affected and the detection accuracy remains unchanged.

3.2. Organization of the Detector Hierarchy. In this section, we focus on the organization of the tree-structured detector hierarchy. As mentioned in Figure 4, different levels work on all subwindows in a pipelined fashion, which means that when detector nodes in the second level start processing the jth subwindow, the nodes in the first level can work on the $(j+1)$th subwindow. Also, classifiers for different classes in a single level work in parallel.

Different detector nodes have similar structure with different classification parameters. The input materials of each detector node are some feature values of sub-windows, and an enable signal indicating whether the node needs to be activated. The output of each detector node is a Boolean value indicating whether the subwindow belongs to the current face class. If the node is not a leaf node, the output will be sent into all its child nodes in the next level as their enable signals; otherwise, the output will be the final results of the detector hierarchy. In each node, cascaded structure [4] (see Figure 1) is also adopted to organize strong classifiers in it. The motivation behind the cascade of classifier is that simple classifiers at early stage can filter out most negative subwindows efficiently, avoiding activating classifiers at later stages.

In Figure 6, we pick up several detector nodes from two adjacent levels to explain the structure and their communications. All direct child nodes of a parent node (we call them "brother nodes") uses the same feature set. The reason is that they share the same selected features while training the first stage of weak classifiers (see Algorithm 3). Therefore, the feature values calculated by the Feature Calculator are sent into all direct nodes of the same parent node simultaneously. When a subwindow passes into the parent level in Figure 6, if the Enable Signal from the parent node of the parent level is negative, the node in the parent level will not be activated, since the subwindow is not considered to belong to this corresponding class; otherwise, the sub-window is processed in *Stage 1* of the parent node. If the output of

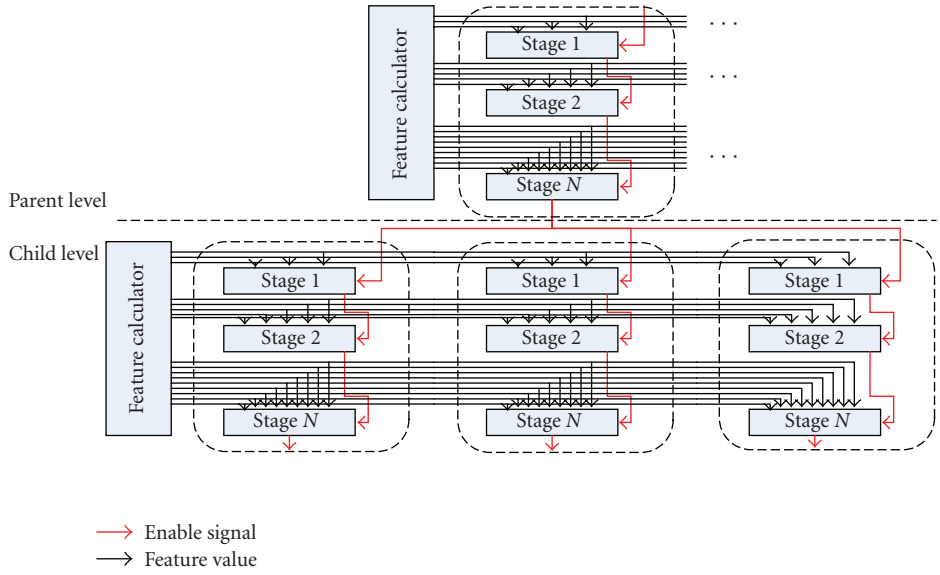

Figure 6: Structure of detector nodes and their communications.

the *Stage 1* is negative, the subwindow does not have to be processed by later stages in the node, which can decrease the computational cost; otherwise, the Enable Signal of the next stage will be positive. The final output of the current node is positive only if the final stage's output is positive. The output of the parent node is connected with the Enable Signals of its direct child nodes. The workflow of these child nodes are the same with that of the parent node. We have mentioned that different levels work on all sub-windows in a pipelined fashion. Actually, different stages in a level form a pipeline as well.

3.3. Parallel Implementation of the Strong Classifier. This section will describe the structure of the strong classifiers in each cascade stage of each detector node. The design of the strong classifier has been proposed in Section 2.4. For each level in each direct child node of a parent detector node, the strong decision function is a particular sum of products, where each product is made of a constant α_t and the value -1 or $+1$ depending on the output of h_t.

Different strong classifiers have similar structure with different classification parameters and different number of weak classifiers. For strong classifiers in different stages of a detector node, the number of weak classifiers is different. In addition, the constant values α_t and the feature set used by weak classifiers are different too. However, the constant values α_t and the feature set are the same for the same cascade stage of different direct nodes of a parent node according to the training procedure in Algorithm 2.

We illustrate a possible structure of the strong classifier in Figure 7. However, T feature values calculated by the Feature Calculator are sent into T weak classifiers h_t. Due to the binary nature of the output of h_t, it is possible to avoid computation of multiplications. We can simply use the output of h_t to determine the sign of α_t through a multiplexer. The results of T multiplexers are summed by a hierarchy of adders. We can see that T-1 addition operations are required and the latency will be T-1 for a sequential pipeline and $\log_2 T$ for a tree-structured pipeline, which will slow down the processing speed when T is large.

We can improve the above structure by decreasing the addition operations in the hierarchy of adders. The idea is that the multiplexers can be replaced by lookup table units, and the results of additions and subtractions of α_t are encoded into them beforehand. The outputs of the weak classifiers are used as addresses of LUT units. The more additions and subtractions are encoded in a single LUT unit, the less online addition operations are required in the hierarchy of adders. The improved structure of the strong classifier with 16-bit LUT unit is presented in Figure 8.

3.4. Design of the Weak Decision Function. In this section, we will introduce the hardware structures of the two-stage weak classifiers. The design of the weak classifiers has been discussed in Section 2.5. The first stage $\mathbf{h}_t' : \boldsymbol{\chi} \to \mathbb{R}^k$ has the following formula: $\mathbf{h}(x; \theta_k, \mu_k) = \sum_{j=1}^{p} c_j^* B_p^j(\theta_k)$, where

$$B_p^j(u) = \begin{cases} 1 & u \in \left[\dfrac{(j-1)}{p}, \dfrac{j}{p}\right) \\ 0 & u \notin \left[\dfrac{(j-1)}{p}, \dfrac{j}{p}\right) \end{cases}, \quad j = 1, \ldots, p, \tag{6}$$

(see Algorithm 3) and c_j^* is trained offline. The second stage $\mathbf{h}_t'' : \mathbb{R}^k \to \{-1 \mid +1\}^k$ has the form of (4) for each dimension of the k classes.

The first and the second stages are combined to form the weak classifier h_t required by the strong classifier as shown in Figure 8. For the tth weak classifier, the feature value is sent into all dimensions of $\mathbf{h}_t'$ first, where each dimension corresponds to each direct child node of a parent detector node. Following the decision procedure based on piece-wise function, all k outputs of $\mathbf{h}_t'$ are sent into each dimension

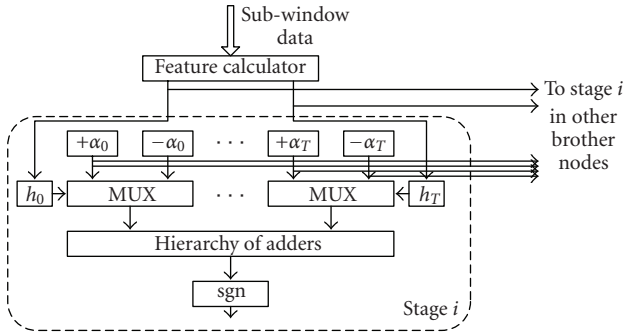

Figure 7: A possible structure of the strong classifier.

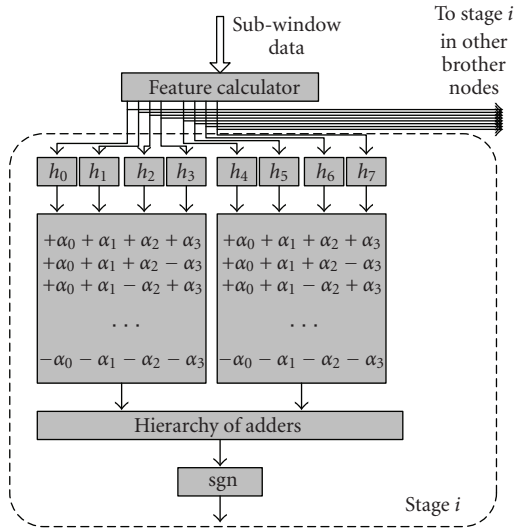

Figure 8: Improved structure of the strong classifier.

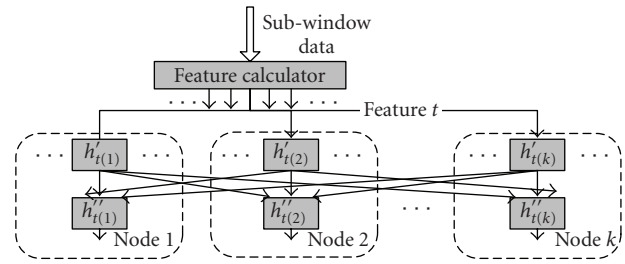

Figure 9: Dataflow of the weak decision function.

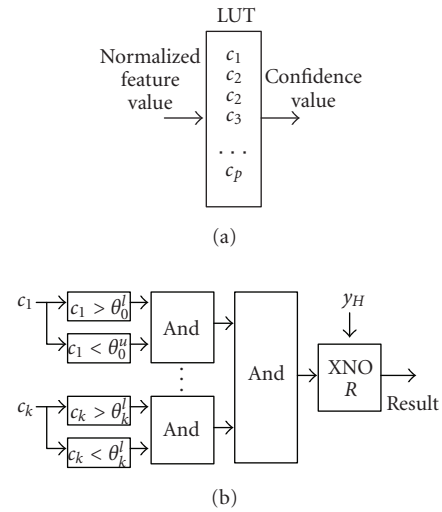

Figure 10: Structures of the two-stage weak classifiers. (a) the first stage; (b) the second stage.

of $\mathbf{h}_t^{''}$ for further hyper-rectangle-based decision. Figure 9 shows the dataflow.

Different $\mathbf{h}_t^{'}$ have similar structure. According to the training procedure in Algorithm 3, the division of feature space and the feature value as input is the same for different dimensions of $\mathbf{h}_t^{'}$, but the constant c are different, which will eventually affect the outputs of different dimensions. The derivation of the confidence values, $c_1, c_2, \ldots, c_p$, is based on the training procedure in Algorithm 3. For all k dimensions of a set of first-stage weak classifiers $\mathbf{h}_t^{'} = \{h^{'}{}_{t(i)} \mid i = 1, \ldots, k\}$ (as shown in Figure 9), the determination of their confidence values uses the same set of training samples. Therefore, the jth confidence value for all k dimensions can form a confidence vector $\mathbf{c}_j$. Also, $\mathbf{c}_j$ is updated iteratively by using a proper optimization algorithm such as Newton-step method, until the training loss $\sum_{S_j} D_t(i) \exp(-y_i(\boldsymbol{\nu}_i \cdot \mathbf{c}_j))$ is minimized. The final $\mathbf{c}_j$, denoted as $\mathbf{c}_j^*$, is then used to construct the piece-wise decision function $\mathbf{h}_t^{'}$. The piece-wise decision function can be easily implemented on hardware by using LUT units, as shown in Figure 10(a). Each element in an LUT unit corresponds to the constant value for a feature bin. The input is the normalized feature value, which will be used as the address for LUT. If the feature value is located at the jth bin of all p bins, c_j in the LUT will be selected as output.

Different $\mathbf{h}_t^{''}$ have similar structure, too. The confidence values as input are the same for different dimensions of $\mathbf{h}_t^{''}$. However, the classification parameters as shown in (3) are different. The hyper-rectangle-based decision function can be easily implemented on hardware by only using some comparison units and logical operation units, as illustrated in Figure 10(b).

4. Reconfiguration of the Architecture Model

In this section, we focus on finding automatically an appropriate tradeoff among the hardware implementation cost, the detection accuracy, and speed by dynamically reconfiguring the hardware architecture model, so that the proposed design can easily meet the demands of different applications. The design automation strategies are analyzed on different design levels from the global structure to the weak classifiers.

4.1. Reconfiguration on the Global Structure Level. In this section, we analyze how the variation of the scaling factor and the scanning step would influence the classification speed and accuracy. Obviously, the larger the scaling factor and the scanning step is, the fewer subwindows are to be processed for a single frame, which will improve the speed but decrease

the accuracy. Therefore, we can find a tradeoff between the speed and accuracy by changing the scaling factor and the scanning step.

By rescaling the integral image, a pyramid framework will be generated. The calculation of pyramid poses heavy computation load. For example, given an input image resolution of x-by-y, and the size of the sub-window for classification x_0-by-y_0, let the scaling factor be r and let S_x and S_y be the steps in the x- and y-direction, respectively. The total number of subwindows, n, can be calculated as

$$n \leq \frac{xy}{S_x S_y} \frac{r^2 - \min(x_0^2/x^2, y_0^2/y^2)}{r^2 - 1}. \tag{7}$$

Typical values of x_0 and y_0 is 24, and the values of S_x and S_y are in the range of 1 to 5 pixels. When $r = 1.2$ and input image size is 256×256, according to (7), n is more than 100 000 images.

The execution time of a single frame depends directly on the number of subwindows n. Therefore, when the system requires that the classification should be processed in a very high speed and the accuracy requirement is not that critical, we can achieve this by increasing r, S_x and/or S_y. Otherwise, we can decrease them.

Let l_{T1}, l_{T2}, and l_{T3} denote the computation latency for a single subwindow for Task 1 (integral image computation), Task 2 (classifier), and Task 3 (image rescaler). According to the pipeline scheme in Figure 4, the latency for generating a final result of a subwindow would approximately depend on the largest value among l_{T1}, l_{T2}, and l_{T3}. Therefore, the total time T_F to process a single frame (with n subwindows) is approximately given by

$$T_F = n \cdot \max(l_{T1}, l_{T2}, l_{T3}). \tag{8}$$

Let c_{T1}, c_{T2}, and c_{T3} denote the hardware implementation costs of Task 1, Task 2, and Task 3, respectively. Then the total cost for the key part of the system is approximately the sum of them:

$$c_{\text{total}} = c_{T1} + c_{T2} + c_{T3}. \tag{9}$$

4.2. Reconfiguration on the Detector Hierarchy Level. In the detector hierarchy level, we analyze two aspects of reconfiguration problems. Firstly, the detection granularity and range of detector nodes in the tree-structured detector hierarchy can be reconfigured. Secondly, the number of cascaded strong classifiers can be reconfigured in each detector node.

4.2.1. Reconfiguration of the Detection Granularity and Range. According to Section 2.3, the proposed tree structure can cover all ± 90-degree ROP and 360° RIP pose changes. The finer the detection granularity in each pose dimension is, the more accurate the detection is. For example, in Figure 2, the granularity in ROP and RIP is 20° and 30°, respectively. Consequently, the hardware resource requirements and the power consumption will increase because of the increase of number of detector nodes. Also, the detection speed may decrease.

For different classification granularities, the detector nodes should be retrained. The consequences are that the number of cascades in the nodes and the number of weak classifiers in the strong classifiers would be totally different. As a result, the reconfiguration of the classification granularity means that almost all detector nodes in the tree should be removed and new detector nodes are configured into the FPGA chip. The cost of this kind of reconfiguration would be a little large. Therefore, it is not so suitable for run-time reconfiguration. However, when the system is about to be used for another purpose which has different requirements on the detection granularity and hardware resources, the reconfiguration operation can be performed during the initialization phase. By reconfiguring the current system, there is no need to design a new system for the application.

Moreover, if the application only requires that the detection be performed in some particular ranges of pose changes, but not all ± 90° ROP and 360° RIP pose changes, we can achieve this by manually shut off those absolutely useless nodes. For example, if pose changes in the range of $(-30°, 30°)$ for ROP are required to be classified, and pose changes in other ranges are not our interest and considered to be nonface directly, the corresponding detector nodes can be manually shut off. In this way, the active nodes are significantly reduced from 41 to 15. The cost of this kind of reconfiguration is small since only a 1-bit enable signal is required for each node and the node itself do not have to be replaced with a new one.

The hardware implementation cost of the detector hierarchy c_{T2} can be approximately calculated as

$$c_{T2} = \sum_{i=1}^{m} \sum_{j=1}^{n_i} c_{ij}, \tag{10}$$

where c_{ij} is the cost of the jth detector node in the ith detector level.

4.2.2. Reconfiguration of the Cascade Number. As mentioned in Section 3.2, the cascaded structure [4] is adopted in each detector node. As discussed in [4], simple classifiers at early stages can filter out most negative subwindows efficiently, and stronger classifiers at later stages are only necessary to deal with instances that look like faces with some particular poses. Compared with the detector node with only one strong classifier, the power consumption will be significantly decreased since quite a few subwindows can be processed with the early simple classifiers. However, it is at the cost of hardware resources. The approximate cost of the cascaded detector node node_{ij} $(i = 1, \ldots, m, j = 1, \ldots, n_i)$ is

$$c_{ij} = \sum_{k=1}^{q_{ij}} c_{ijk}, \tag{11}$$

where c_{ijk} is the estimated hardware cost of the kth cascade stage at the jth node in the ith level.

The number of the cascades in a node can be reconfigured to meet different requirements. For applications

which demand that the power consumption be low, such as digital camera and other handheld devices, more cascades can be configured into the FPGA chip; otherwise, for applications which demand that the size and price of the device be controlled, the number of the cascades should be constrained.

4.3. Reconfiguration on the Strong Classifier Level. The number of weak classifiers, denoted as T_{ijk}, in a single strong classifier (we are referring to the strong classifier at the latest stage in a detector node) can also be reconfigured to find a tradeoff between the detection accuracy and hardware implementation cost. If T_{ijk} is smaller, the accuracy is weaker and the hardware cost is less; otherwise, the accuracy is stronger and the cost is more.

The hardware implementation cost of a strong classifier with T_{ijk} weak classifiers can be approximately estimated. According to Figures 8 and 9, a strong classifier is mainly composed of a set of 16-bit LUT units, a hierarchy of adders and T_{ijk} weak classifiers containing the first stage h'_t, the second stage h''_t together with their corresponding feature calculation units. In terms of slices, the hardware cost, c_{ijk}, for the strong classifier at kth cascade stage of jth detector node at ith detector level, can be expressed as follows:

$$c_{ijk} = \sum_{t=1}^{T_{ijk}} \left(c_f^t + c_{h'}^t + c_{h''}^t\right) + \left\lceil \frac{T_{ijk}}{4} \right\rceil \cdot c_{LUT} + \left(\left\lceil \frac{T_{ijk}}{4} \right\rceil - 1\right) \cdot c_{\text{add}}, \tag{12}$$

where c_f^t, $c_{h'}^t$, $c_{h''}^t$, c_{LUT}, and c_{add} are the costs of a feature calculation unit, a first-stage weak classifier h'_t, a second-stage weak classifier h''_t, a 16-bit LUT unit, and an adder, where c_{LUT} and c_{add} will be considered as a constant.

4.4. Reconfiguration on the Weak Decision Function Level. In this section, the reconfiguration problems on the first-stage weak classifier h'_t and second-stage h''_t are analyzed.

As shown in Figure 10(a), the elementary structure of h'_t is an LUT unit. The number of elements in the LUT unit is the same with the division granularity of feature space. The finer the granularity is, the more accurately the function can estimate. However, more LUT elements will cost more hardware resources. Therefore, we can tune the number of elements to meet different requirements on the accuracy and the cost. The hardware implementation cost of the first-stage weak classifier, $c_{h'}$, is approximately proportional to the number of LUT elements u:

$$c_{h'} = u\lambda_1, \tag{13}$$

where λ_1 is the cost for one LUT element.

As the elementary structure of h''_t is based on numerous comparisons performed in parallel (see Figure 10(b)), it is necessary to reduce the number of comparators as much as possible for minimizing the final number of used slices. If the hardware resources are limited, three particular cases of hyper-rectangles can be considered to replace the general case as illustrated in (4) for minimizing the hardware cost.

(1) The single threshold:

$$\mathbf{T} = \{\theta_i, y_T\}, \tag{14}$$

where θ_i is a single threshold, $i = 1, \ldots, k$ (k is the number of classes in the classification space), and the decision function is

$$h_T(x) = y_T \Longleftrightarrow c_i < \theta_i, \quad h_T(x) = -y_T, \text{otherwise.} \tag{15}$$

(2) The single interval:

$$\mathbf{I} = \left\{\theta_i^l, \theta_i^u, y_I\right\}, \tag{16}$$

where the decision function is

$$h_I(x) = y_I \Longleftrightarrow \left(c_i > \theta_i^l\right) \text{ and } \left(c_i < \theta_i^u\right), \quad h_I(x) = -y_I, \text{otherwise.} \tag{17}$$

(3) The partial hyper-rectangle:

$$\mathbf{P} = \left\{\theta_1^l, \theta_1^u, \theta_2^l, \theta_1^u, \ldots, \theta_p^l, \theta_1^u, y_P\right\}, \tag{18}$$

where $\{\theta_1^l, \theta_1^u, \theta_2^l, \theta_1^u, \ldots, \theta_p^l, \theta_1^u\}$ are the lower and upper limits of a randomly selected subset of the entire k-dimensional space. And the decision function is the similar with (4):

$$h_P(x) = y_P \Longleftrightarrow \prod_{i=1}^{p} \left(\left(c_i > \theta_i^l\right) \text{ and } \left(c_i < \theta_i^u\right)\right), \quad h_P(x) = -y_P, \text{otherwise.} \tag{19}$$

In order to optimize the final result, it is necessary to combine the previous approaches, finding for each iteration the best weak classifier among the single threshold h_T, the single interval h_I, the partial hyper-rectangle h_P, and the general hyper-rectangle h_H. The training procedure of the second-stage weak classifier h''_t can be optimized as shown in Algorithm 4.

This strategy allows minimizing the number of training iterations, and thus minimizing the final hardware cost. The hardware implementation cost of a second-stage weak classifier, $c_{h''}$, is approximately proportional to the number of comparators v:

$$c_{h''} = v\lambda_2, \tag{20}$$

where λ_2 is the cost for one comparator.

(1) Train classifier with respect to the weighted samples set and obtain some hypothesis $h_T, h_I, h_P{}^{(1)}, \ldots, h_P{}^{(d)}$ and h_H;
(2) Calculate weighted errors $\varepsilon_T, \varepsilon_I, \varepsilon_P{}^{(1)}, \ldots, \varepsilon_P{}^{(d)}$ and ε_H introduced by each classifier;
(3) Choose h_t'' from $\{h_T, h_I, h_P{}^{(1)}, \ldots, h_P{}^{(d)}, h_H\}$ for which $\varepsilon_t = \min(\varepsilon_T, \varepsilon_I, \varepsilon_P{}^{(1)}, \ldots, \varepsilon_P{}^{(d)}, \varepsilon_H)$.

Algorithm 4: Optimization for the training procedure of the second-stage weak classifier.

4.5. Design Space Exploration Algorithm. Based on the above analysis, a design space exploration algorithm is proposed so that the reconfiguration strategies on all design levels can be well coordinated to meet the demands on the hardware implementation cost, the detection accuracy and speed for different applications.

To describe the algorithm more clearly, we first list some related parameters in Table 1.

The design space exploration algorithm is illustrated in Algorithm 5. At first, the related parameters are initialized without considering the hardware resources constraints in order to increase the detection speed and accuracy as possible as we can. Then the approximate hardware cost and execution time for a single frame is estimated based on the parameters and the architecture model. If the available hardware resources are sufficient and the processing speed is fast enough, we activate the system to test the detection accuracy. If the error rate is ideal enough, the algorithm returns. If the error rate does not meet the requirements, some parameters are demanded to be tuned. There are two considerations about decreasing the error rate: refining the scanning granularity and improving the classification accuracy of detector nodes. The scanning steps in the x- and y-direction S_x, S_y, and the scaling factor r can be decreased to increase number of subwindows n based on (7), so as to refine the scanning granularity and avoid skipping some potential face objects. As for improving the classification accuracy, the number of weak classifiers in each strong classifier T_{ijk} can be increased by doing more training iterations as shown in Algorithm 2. Also, the number of LUT elements in each first-stage weak classifier u can be increased via finer division of feature space during the training procedure. Also, the number of comparators in each second-stage weak classifier v can also be tuned by using the training procedure as shown in Algorithm 4. After the above parameters are updated, the algorithm starts a new iteration to check whether the hardware implementation cost, the detection accuracy and speed meet the requirements or not. If the error rate is ideal enough, but the processing speed is not fast enough, the scanning granularity should be made coarser and the computational complexity of detector nodes should be decreased, which are opposite to the update process for decreasing the error rate. Therefore, the parameters S_x, S_y, r, T_{ijk}, u, and v should be updated in an opposite way correspondingly. If the error rate and the processing speed both meet the requirements, but the hardware implementation cost is not ideal, the number of cascade stages in each detector node q_{ij}, in addition to T_{ijk}, u, and v can be decreased to reduce the hardware cost, which will sacrifice some detecting accuracy inevitably.

Table 1: List of related parameters.

Symbols	Description
x, y	the input image resolution
x_0, y_0	the size of a sub-window
c_{avail}	the available hardware resources on the FPGA chip for the key part of the system
c_{total}	the estimated total hardware cost for the key part of the system
T_{goal}	the expected execution time for a frame
T_{F}	the estimated total execution time for a frame
n	the number of sub-windows to be processed in a frame
c_{T1}, c_{T2}, c_{T3}	the estimated hardware implementation costs of Task 1, Task 2 and Task 3
l_{T1}, l_{T2}, l_{T3}	the computation latency for a single sub-window for Task 1, Task 2 and Task 3
S_x, S_y	the scanning steps in the x- and y-direction
r	the scaling factor
m	the number of levels in the detector tree
n_i	the number of nodes at the ith detector level
q_{ij}	the number of cascades at the jth node in the ith level
T_{ijk}	the number of weak classifiers in a strong classifier
c_{ij}	the estimated hardware cost of at the jth node in the ith level
c_{ijk}	the estimated hardware cost of the kth cascade at the jth node in the ith level
c_f^t	the hardware cost of a feature calculation unit
$c_{h'}{}^t$	the hardware cost of a first-stage weak classifier
$c_{h''}{}^t$	the hardware cost of a second-stage weak classifier
c_{LUT}	the hardware cost of a 16-bit LUT unit
c_{add}	the hardware cost of an adder
λ_1	the hardware cost of a LUT element in a first-stage weak classifier
u	the number of LUT elements in a first-stage weak classifier
λ_2	the hardware cost of a comparator in a second-stage weak classifier
v	the number of comparators in a second-stage weak classifier

The parameters that should be initialized include S_x, S_y, r, m, n_i, q_{ij}, T_{ijk}, u, v. The scanning steps S_x and S_y can be initialized as small as 1. The scaling factor r is a value larger than 1, such as 1.25. The number of detector levels m is fixed to be 5 in our work. The number of nodes at the ith detector

```
INPUT: x, y, x0, y0, c_avail, T_goal, c_T1, c_T3, l_T1, l_T2, l_T3, c_f^t, c_LUT, c_add, λ1, λ2, and the initial values for S_x, S_y, r, m, n_i,
q_ij, T_ijk, u, v;
OUTPUT: the updated values for S_x, S_y, r, q_ij, T_ijk, u, v;
BEGIN
    num_iterations = 0;
    Initialize S_x, S_y, r, m, n_i, q_ij, T_ijk, u, v without considering the hardware resources constraints;
tag1: IF (num_iterations>THRES) FAIL;        //if this algorithm is iterated THRES times, the algorithm stops.
    Estimate c_h' and c_h'' using Equations (13) and (20); Estimate c_ijk using Equation (12); Estimate c_ij using
Equation (11); Estimate c_T2 using Equation (10);
    Calculate c_total using Equation (9);
    IF (c_total <= c_avail)    //the hardware resources are sufficient
    BEGIN
        Calculate n using Equation (7);
        Estimate T_F using Equation (8);
        IF (T_F <= T_goal)        //the processing speed is fast enough
        BEGIN
            Configure the system architecture model using the related parameters;
            Activate the system to run RIMVFD tasks on some test images;
            IF(the error rate is below the predetermined threshold)            RETURN;
            ELSE BEGIN
                Update s_x, s_y, r to increase n, and/or train the classifiers using updated T_ijk, u, v;
                num_iterations + = 1;
                GOTO tag1;
            END
        END
        ELSE BEGIN
            Update s_x, s_y, r to decrease n, and/or train the classifiers using updated T_ijk, u, v;
            num_iterations + = 1;
            GOTO tag1;
        END
    END
    ELSE BEGIN
        IF (there exists q_ij which are larger than 1)    //reduce the hardware cost by reducing the number
        of cascade stages
        BEGIN
            Reduce the number of cascade stages in each detector node;
            num_iterations + = 1;
            GOTO tag1;
        END
        Train the classifiers using updated T_ijk, u, v;        //reduce the hardware cost by sacrificing some
    detecing accuracy
        num_iterations + = 1;
        GOTO tag1;
    END
END
```

Algorithm 5: The design space exploration algorithm.

level, n_i, is fixed to be 1, 1, 3, 9, 27, respectively for $i = 1, 2, \ldots, 5$. Therefore, the total number of detector nodes is 41, and these nodes are organized as described in Section 3.1. The number of stages in the cascade construction of each detector node, q_{ij}, and the number of weak classifiers in each cascade stage, T_{ijk}, are determined by selecting operating points within a receiver operator characteristic (ROC) curve. In each first-stage weak classifier, the number of LUTs, u, is initialized as 64. Also, the number of comparators in a second-stage weak classifier v is initialized to be $2k$ according to (4), where k is the face class dimensions of the current level. During the design space exploration, S_x, S_y, r, q_{ij}, T_{ijk}, u, v are updated iteratively. While increasing or decreasing S_x, S_y, r to tune the scanning granularity, the updating steps can be 1, 1, and 0.05 respectively. For q_{ij}, as long as q_{ij} is larger than 1, q_{ij} can be decreased with a step of 1 for reducing the hardware cost. For T_{ijk}, the tuning step is set to be 10 here. Also, the step is 4 for u. v is increased or decreased between 1 and $2k$ with a step of 1.

To sum up, if any one of the hardware implementation cost, the detection accuracy, and speed do not meet the requirements, the related parameters are tuned and the architecture model is reconfigured with these new parameters. The order of priorities of the cost, speed and accuracy is

Figure 11: Some training samples.

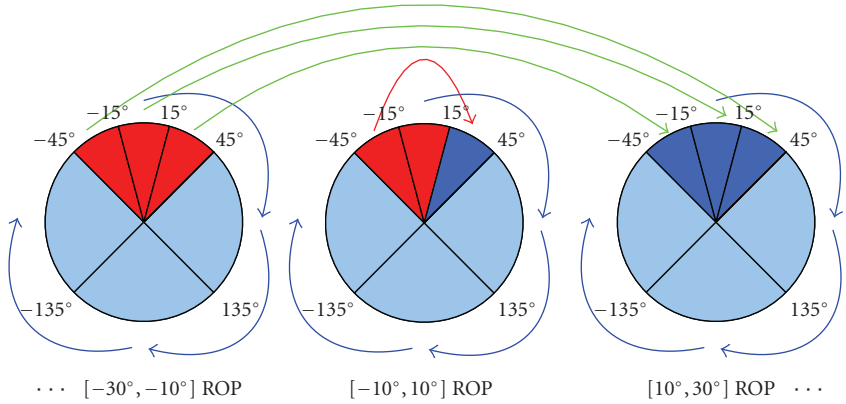

Figure 12: Flip and rotate detectors. (The original detectors of frontal view ([−10°, 10°] ROP) are the [−45°, 15°] and [−15°, 15°] RIP ones. The original detectors of nonfrontal faces are the [−45°, 15°], [−15°, 15°] and [15°, 45°] RIP ones for each ROP view).

cost > *speed* > *accuracy*. The algorithm iterates until the termination condition is satisfied.

5. Experimental Results

In this section, several experiments are performed to evaluate the proposed RIMVFD method and hardware architecture. The training data preparation, training procedure, and the performance analysis of the detection procedure are described.

5.1. Training Dataset. More than 85 000 face samples are collected by cropping from various sources. All the samples are normalized to the standard 24 × 24 pixel patch. The views of the samples cover all ±90° ROP and ±45° RIP pose changes. We partition the sample space into smaller and smaller subspaces of narrower view ranges. At the top level in Figure 2, there is only one detector node. So all face samples are grouped into one class. At the second level, there are four detectors. Since the face samples cover ±45° RIP pose changes, they are used to train the first node in this level. As described in Section 3.1, the other three detector nodes can be generated by simply rotating the first node for 90°, 180°, and 270° clockwise. At the third level, the face samples are grouped into three view classes of (−90°, −30°), [−30°, 30°] and [30°, 90°]. The partition of the samples works in the similar way at later levels. Figure 11 gives some training samples.

5.2. Training Procedure. By using the training method introduced in Sections 2.4 and 2.5, each detector node is trained with the corresponding subset of face samples. Finally, the detector tree is composed of 41 nodes in 5 levels. Because of hardware resources restrictions, we constrain the cascade construction for each detector node to a maximum of 5 stages.

Since any detector can be rotated 90°, any detector node in the (45°, 135°), (135°, −135°), and (−135°, −45°) RIP ranges can be generated by rotating the corresponding detector node in the (−45°, 45°) RIP range for 90°, 180°, and 270°, which reduces 75% of the cost for the training procedure. In addition, due to the horizontal symmetry between the left and right faces, only one side of the detector nodes needs to be trained, and the nodes for detecting faces on the other side can be generated by mirroring the trained nodes. Figure 12 illustrates the scheme of flipping and rotating detectors. In this way, only the detector nodes for the ranges marked as red sectors in Figure 12 should be trained. Finally, the number of detector nodes to be trained is reduced from 161 to 23.

5.3. Performance Evaluation of the Hardware RIMVFD System

5.3.1. Test-Bed and Test Data. In this paper, a prototype of the proposed design has been implemented in an FPGA test-bed for evaluation. The FPGA test-bed includes a large capacity FPGA chip and an SDRAM module, connecting to the host processor through USB interface. Our target device is Altera Stratix II EP2S130F1020C5. This FPGA chip contains 106 032 ALUTs and 6 747 840 bits of onchip memory. The SDRAM module with capacity of 1G Bytes is deployed as offchip memory. We use a PC with a Pentium 4 2.8 GHz CPU and a 1 GB memory bank as the host processor in our test-bed, which functions as the Video Capture Device and Display/Store Device as shown in Figure 5. The FPGA implementation is coded with Verilog HDL, simulated with Mentor Graphics ModelSim, and synthesized with Quartus II. Table 2 introduces the specifications of the proposed hardware prototype.

Table 2: Specifications of the proposed hardware prototype

Modules	Specifications
FPGA chip	Altera Stratix II EP2S130F1020C5 (106 032 ALUTs and 6 747 840 bits of on-chip memory)
host processor	Pentium 4 2.8GHz CPU and 1GB DDR RAM
offchip memory	1 GB
total onchip memory bits	245696 bits
total logic cells	105127 ALUTs
working frequency	98 MHz

Figure 13: Some detection results of the proposed hardware RIMVFD system.

Table 3: Parameter settings for test.

Parameters	Values
x_0	24
y_0	24
S_x	1
S_y	1
r	1.25
u	64
number of detector nodes in the detector tree	41
total number of weak classifiers	3820

All initial image frames are captured by the PC. The processing modules integrated in the FPGA chip read the image data and generate the detection results. The final results are transferred to host processor for displaying and storing.

To give an overall evaluation, we test our MVFD on the CMU profile set, which consists of 208 images with 441 faces. Since the CMU profile set cannot evaluate the rotation invariant characteristics well, we also collect a database of 360° rotation invariant test images from various sources, containing 213 images with 682 rotation invariant multi-view faces. The CMU + MIT frontal face test set containing 130 images with 507 main frontal faces is also used to evaluate the frontal face detection performance. Some of the detection results generated by the proposed hardware RIMVFD system are shown in Figure 13.

5.3.2. Resource Utilization. We synthesize the design on the FPGA using Quartus II. In this experiment, the size of the subwindow for classification, x_0-by-y_0, is 24×24. The scanning steps in the x- and y-direction S_x and S_y are set to be 1 pixel, and the scaling factor r is 1.25. The number of detector nodes is 41, and these nodes are organized as described in Section 3.1. The number of stages in the cascade construction of each detector node, q_{ij}, and the number of weak classifiers in each cascade stage, T_{ijk}, are determined by selecting operating points within a receiver operator characteristic (ROC) curve. In each first-stage weak classifier, the number of LUTs, u, is 64. Finally, 3820 weak classifiers are included in the detector tree. Table 3 lists settings for some related parameters in our experiments.

Based on the synthesis results, the clock speed of the entire design reaches 98 MHz, which can still be increased through P&R (Place and Route) optimization. Table 4 gives the resource utilization for different parts of the system. It can be seen from the synthesis results that the Face Detector Tree occupies most of the ALUT resources. The reason is that the number of weak classifiers in it is pretty large.

5.3.3. Speed Comparison with Software Solution on PC and Related Works. In this section, we compare the processing speed of the hardware implementation with that of the software solution on PC with a Pentium IV processor operating at 2.8 GHz and 1 GB memory bank. The software solution is implemented based on OpenCV. The program code is written in Visual C++ language under Windows XP OS. For the image rescaling procedure, we choose to rescale the scanning subwindow in the software solution, while rescale the image and fix the size of the subwindow in the hardware solution as explained in Section 3.1. Both the hardware and software implementations use the same test dataset as described in Section 5.3.1. The execution time of an image is calculated from reading image data to generating the final result.

Table 5 gives the comparison of execution latency for detecting rotation invariant multi-view faces in one image frame by using these two solutions and the offchip memory bandwidth requirements. Some test images with different sizes are used. We can see from the results that although the clock speed of our design is only 98 MHz, which is much slower than that of PC (i.e., Pentium 4 2.8 GHz), the processing speed of the hardware solution is still faster than

Table 4: Resource utilization for different parts of the system.

Modules	Area (ALUTs)	Onchip memory (bits)
Video capturer/database interface	238	67239
Results display/database interface	322	8325
Main memory controller	819	19200
Integral image computer	657	48921
Face detector tree	102512	69038
Image rescaler	579	32973
Total	105127	245696

Table 5: Performance comparison between the hardware and software solution and the offchip memory bandwidth requirements.

Image size	Execution latency (ms)		Speed-up	Offchip memory bandwidth requirements (MB/s)
	Hardware (98 MHz)	Software (2.8 GHz)		
160×120	1.58	23.2	14.68	99.8
192×144	2.06	33.6	16.31	110.7
320×240	5.36	93.9	17.52	158.7
384×288	7.27	135.4	18.62	168.5
480×320	9.71	188.2	19.38	175.4
640×480	18.42	376.6	20.45	185.1
768×576	26.08	542.5	20.8	188.3
800×600	28.22	588.7	20.86	188.9

that of the software solution. The execution time speedups range from 14.68 to 20.86 for images with size of 160×120 up to 800×600. For the 320×240 video sequences, the frame rate can reach 186 fps, which is 17.52 times faster than the software solution as long as the capture device or the video database can generate the original images fast enough. Even for the 800×600 images, the frame rate can be 35 fps. The offchip memory bandwidth requirements for different image sizes are also described in Table 5. The available offchip memory bandwidth in our prototype system is about 200 MB/s, which can meet the requirements.

In order to measure the latency caused by the second stage of two-stage weak classifiers, the processing speeds for FPGA design with and without two-stage mechanism are compared. For the design without two-stage mechanism, the work flow is similar with that of Huang's Vector Boosting [9]. That is, the confidence values generated in the first stage of weak classifiers are directly used to generate the strong classifier, and the output of the strong classifier is compared with a threshold. Therefore, the length of the entire detector tree pipeline in the design without two-stage mechanism is shorter than that in the design with two-stage mechanism. However, the latency for processing a single subwindow in a frame is almost the same for the pipelines with or without two-stage mechanism. Therefore, the execution latency for processing a frame with a lot of subwindows will be hardly affected by the increasing of the pipeline length. Consequently, the introduction of the two-stage mechanism will not cause too much negative impact on the frame rate. Table 6 shows the performance comparison for FPGA design with and without two-stage mechanism.

We also compare the processing speed between our design and other related works, that is, Rowley's ANN method [2], Viola's cascade detector [4], Schneiderman's Bayesian decision rule method [3], Wu's parallel cascade method [6], and Huang's WFS tree method [9]. In Rowley's work [2], he claimed that a fast version of his system can process a 320×240 pixel image in two to four seconds on a 200 MHz R4400 SGI Indigo 2. Also, about 5 seconds are required to process a 320×240 image in [3]. In Viola's contribution [4], on a 700 MHz Pentium III processor, the face detector can process a 384 by 288 pixel image in about 0.067 seconds, which is roughly 15 times faster than Rowley's detector [2]. The above contributions all focus on frontal face detection. According to the results presented in Table 5, we can see that our work takes about 7.27 milliseconds for RIMVFD on a 384×288 image, which is even faster than frontal face detection in the above contributions. In Wu's work, the running time is about 18 milliseconds for frontal face detection, about 80 milliseconds for MVFD, and about 250 milliseconds for RIMVFD while processing a 320×240 image on a Pentium 4 2.4 GHz PC. Also, Huang reported in [9] that on a Pentium 4 2.8 GHz PC, MVFD takes about 40 milliseconds on the detection of a 320×240 image, and RIMVFD speed is about 11 fps. Obviously, the processing speed in our work is much faster than these related works, which is about 186 fps for 320×240 images. Table 7 lists the comparison results.

The reason why our design outperforms related works is that our design uses the tree-structured detector hierarchy with much less detector nodes, and the temporal/spatial parallelism in the algorithm is fully exploited to construct highly parallel and pipelined hardware architecture.

Table 6: Performance comparison for FPGA design with and without two-stage mechanism.

Image size	Execution latency (ms)	
	with two-stage mechanism	without two-stage mechanism
160 × 120	1.58	1.49
192 × 144	2.06	1.98
320 × 240	5.36	5.31
384 × 288	7.27	7.21
480 × 320	9.71	9.67
640 × 480	18.42	18.39
768 × 576	26.08	25.98
800 × 600	28.22	28.17

Table 7: Performance comparison with related works.

Works	Platform	Image size	Detection type	Speed
Rowley's	R4400 SGI Indigo 2 200 MHz	320 × 240	Frontal	2~4 s
Schneiderman's	—	320 × 240	Frontal	5 s
Viola's	Pentium III 700 MHz	384 × 288	Frontal	0.067 s
Wu's	Pentium 4 2.4 GHz	320 × 240	Frontal	18 ms
			MVFD	80 ms
			RIMVFD	250 ms
Huang's	Pentium 4 2.8 GHz	320 × 240	MVFD	40 ms
			RIMVFD	11 fps
Ours	FPGA 98 MHz	384 × 288	RIMVFD	7.27 ms

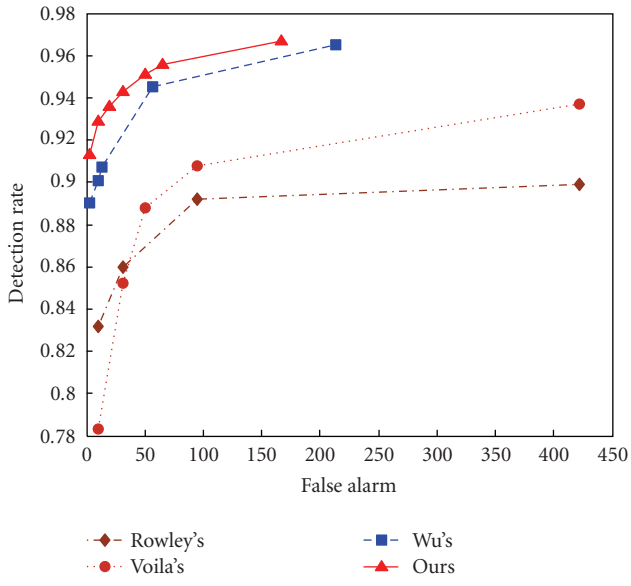

Figure 14: ROC curves on CMU + MIT frontal face set.

5.3.4. Accuracy Comparison with Related Works. To evaluate the detection accuracy of the proposed RIMVFD method, the results are compared with some previous published works, that is, Rowley's ANN method [2] and Viola's cascade detector [4] for frontal face detection, Schneiderman's Bayesian decision rule method [3] for MVFD, Wu's parallel cascade method [6], and Huang's WFS tree method [9] for MVFD.

Frontal Face Detection Test on CMU + MIT Frontal Face Set. As frontal faces are very useful in face-related applications, we first evaluate the proposed method on the CMU + MIT frontal face test set, and compare the results with those of Rowley's ANN method [2], Viola's cascade detector [4], and Wu's parallel cascade method [6]. Figure 14 gives the ROC curves. We can see that our proposal has a much higher detection rate.

MVFD Test on CMU Profile Set. We compare the proposed method with the previous works, that is, Schneiderman's Bayesian decision rule method [3] for MVFD, Wu's parallel cascade method [6] and Huang's WFS tree method [9] for MVFD. All tests are based on the CMU profile set. Li et al. [7] also tested their pyramid detector on this set, but did not report any statistic on the results. Figure 15 gives the ROC curves.

RIMVFD Test on our Own Database. For the rotation invariant MVFD, since there is no available standard test set, we collect a database of 360°-degree rotation invariant test images from various sources, containing 213 images with 682 rotation invariant multi-view faces. In this test set, 57 images with 142 multi-view faces are selected from CMU profile set, and the rest 156 images with 540 faces are captured with digital cameras covering various ROP poses from −90° to 90° , and then manually rotated to get faces with various RIP poses in the entire 360°-degree range. Wu's parallel cascade method [6] and Huang's WFS tree method [9] did some

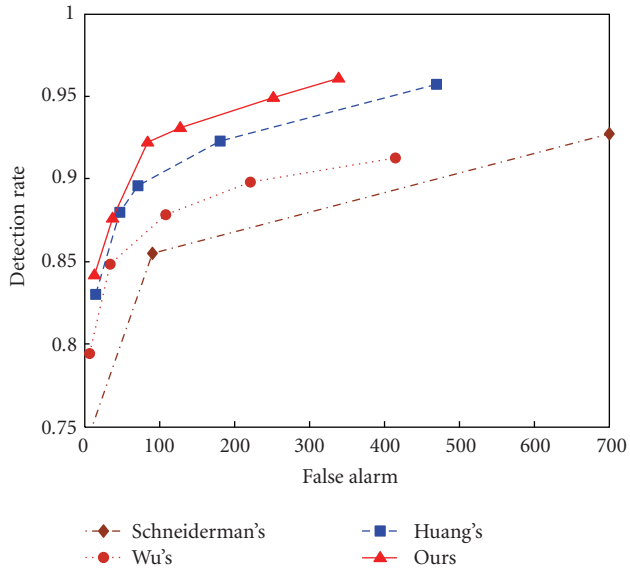

Figure 15: ROC curves on CMU profile set.

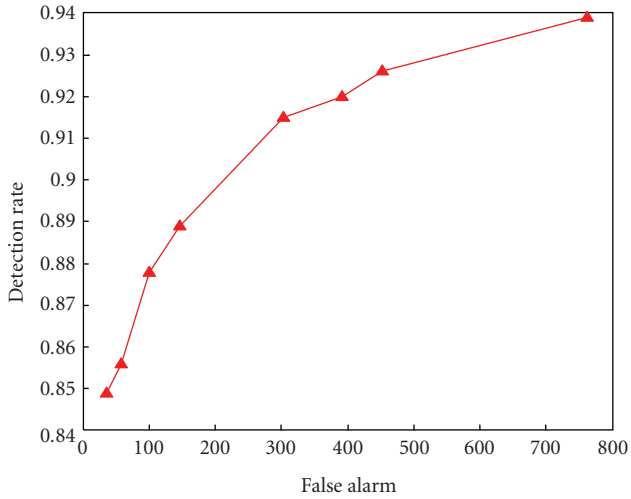

Figure 16: The ROC curve on our own test set.

experiments on RIMVFD, but they did not give the ROC curves. In our experiment, the ROC curve of our proposal on our own test set is given in Figure 16.

The main reason why our proposal achieves a better performance is that the outputs of the first-stage weak classifiers are further classified by the second-stage weak classifiers. Consequently, faces belonging to each class can be well separated from other classes.

5.3.5. Performance Impact of System Reconfiguration. In this section, we evaluate the variation of the hardware implementation cost, detection speed, and accuracy while reconfiguring the proposed RIMVFD architecture model as discussed in Section 4.

Reconfiguring the Scanning Step and Scaling Factor. According to (7), the number of subwindows to be processed is influenced by the scanning step $\{S_x, S_y\}$ and scaling factor

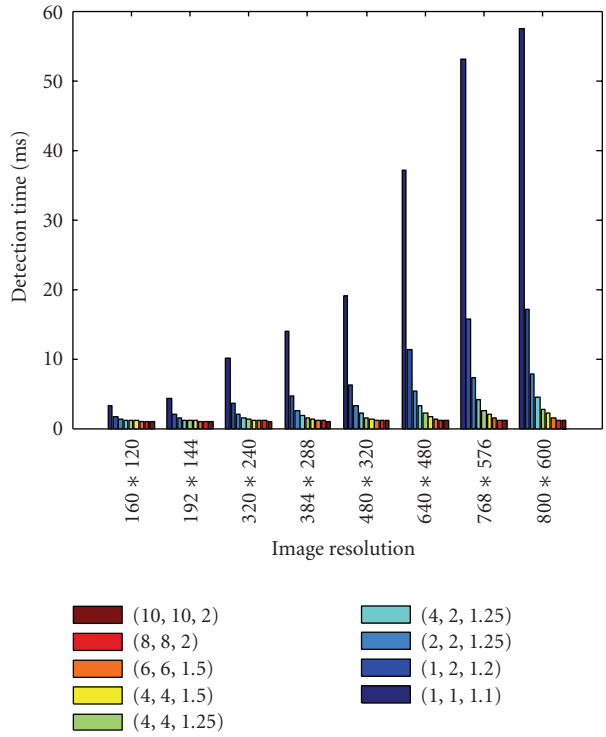

Figure 17: Variation of detection speed while changing S_x, S_y and r.

r. We evaluate the variations of detection speed and accuracy on our own database while changing S_x, S_y and r, as shown in Figures 17 and 18. In this experiment, the other parameters are the same with what have been described in Section 5.3.2.

It can be seen from Figure 17 that the detection speed increases with the increasing of S_x, S_y and r. The reason is that the increasing of S_x, S_y and r will obviously decrease the number of sub-windows in a frame.

From the ROC curves in Figure 18, we can see that the detection rate decreases with the increasing of S_x, S_y and r. The reason is that the increasing of S_x, S_y and r will obviously decrease the number of subwindows in a frame and thus skip some potential faces while scanning the frame.

Reconfiguring the Number of Detector Nodes, Cascade Stages and Weak Classifiers. According to (10), (11), (12), the number of detector nodes, cascade stages in each node and weak classifiers in each strong classifier determines the hardware cost of the detector tree. The sum of stages in all nodes is the number of all strong classifiers in the system actually. As discussed in Section 4.2, we can configure different numbers of nodes and cascade stages in each node to meet the requirements of different applications. In addition, the number of weak classifiers in each strong classifier can also be reconfigured.

Figure 19 gives the variation of ALUTs required by the entire detector tree for different number of weak classifiers. As expected, the number of ALUTs grows approximately linearly with the number of weak classifiers. If the hardware has more resources, more weak classifiers can be deployed to achieve more accurate face detection (based on the analysis

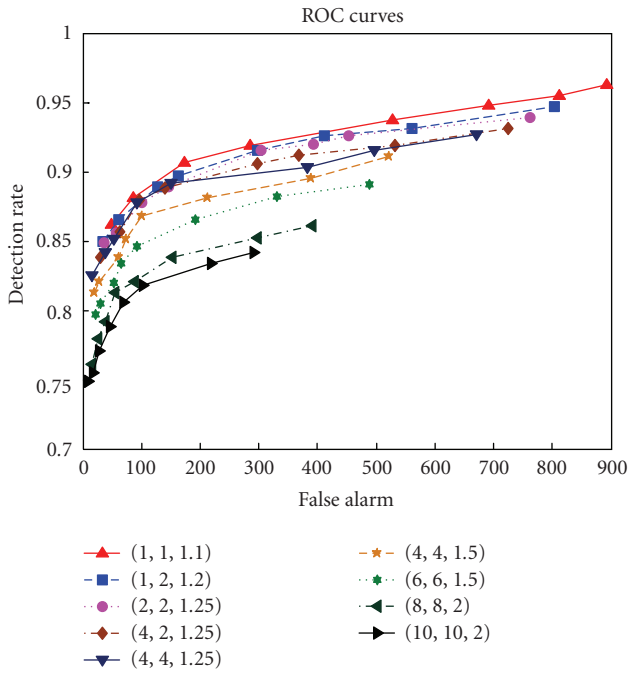

FIGURE 18: Variation of detection accuracy for 320 × 240 images while changing S_x, S_y and r.

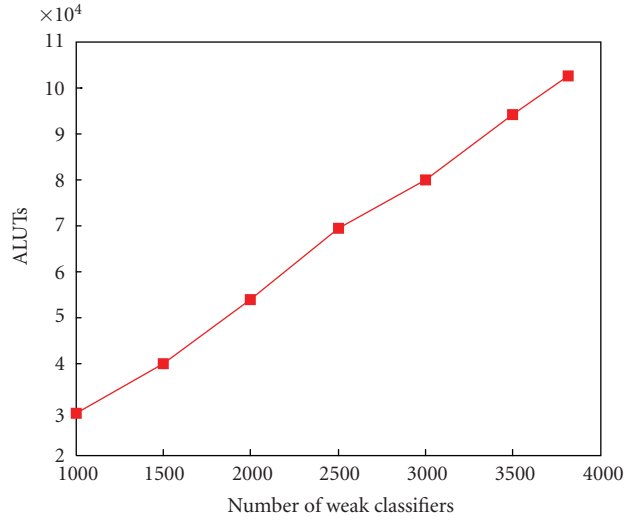

FIGURE 19: Hardware cost variation of the detector tree for different number of weak classifiers.

in [14], the detection error reduces monotonously with the increasing of features (equally with the increasing of weak classifiers) within a certain range), and the detection granularity can be finer to classify the face poses more precisely.

6. Conclusions

For detecting rotation invariant multi-view faces with all −/+90° ROP and 360°-degree RIP pose changes, we presented a fine-classified method and an FPGA-based reconfigurable architecture. A tree-structured detector hierarchy was designed to organize multiple detector nodes. A fine-classified boosting algorithm was proposed to train each detector node, where each weak classifier is a novel two-stage structure. The proposed method achieves higher accuracy than related works. Due to the highly parallel and pipelined design of the hardware architecture for RIMVFD and the reusability of detector nodes, marvelous speed were realized compared with previous related works.

Acknowledgment

This work is supported in part by the National Science Foundation of China through grants 60633050 and 60833004.

References

[1] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, 2002.

[2] H. A. Rowley, *Neural network-based human face detection*, Ph.D. thesis, Carnegie Mellon University, 1999.

[3] H. Schneiderman and T. Kanade, "A statistical method for 3d object detection applied to faces and cars," in *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '00)*, vol. 1, pp. 746–751, Hilton Head Island, SC, USA, 2000.

[4] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '01)*, vol. 1, pp. 511–518, 2001.

[5] A. Kuchinsky, C. Pering, M. L. Creech, D. Freeze, B. Serra, and J. Gwizdka, "FotoFile: a consumer multimedia organization and retrieval system," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 496–503, Pittsburgh, Pa, USA, 1999.

[6] B. Wu, H. Ai, C. Huang, and S. Lao, "Fast rotation invariant multi-view face detection based on real adaboost," in *Proceedings of the 6th IEEE International Conference on Automatic Face and Gesture Recognition (FGR '04)*, pp. 79–84, May 2004.

[7] S. Z. Li, L. Zhu, Z. Q. Zhang, A. Blake, H. J. Zhang, and H. Shum, "Statistical learning of multi-view face detection," in *Proceedings of the 7th European Conference on Computer Vision (ECCV '02)*, pp. 117–121, Copenhagen, Denmark, May 2002.

[8] M. Jones and P. Viola, "Fast multi-view face detection," Tech. Rep. MERL-TR2003-96, Mitsubishi Electric Research Laboratories, 2003.

[9] C. Huang, H. Ai, Y. Li, and S. Lao, "Vector boosting for rotation invariant multi-view face detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV '05)*, vol. 1, pp. 446–453, October 2005.

[10] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[11] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[12] R. Xiao, L. Zhu, and H.-J. Zhang, "Boosting chain learning for object detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV '03)*, vol. 1, pp. 709–715, Nice, France, 2003.

[13] S. Z. Li and Z. Q. Zhang, "FloatBoost learning and statistical face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1112–1123, 2004.

[14] T. Mita, T. Kaneko, and O. Hori, "Joint Haar-like features for face detection," in *Proceedings of the 10th IEEE International Conference on Computer Vision (ICCV '05)*, vol. 2, pp. 1619–1626, Beijing, China, 2005.

[15] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," in *Proceedings of IEEE International Conference on Image Processing (ICIP '02)*, vol. 1, pp. 900–903, 2002.

[16] C. Liu and H.-Y. Shum, "Kullback-Leibler boosting," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03)*, vol. 1, pp. 587–594, June 2003.

[17] S. Baluja, M. Sahami, and H. A. Rowley, "Efficient face orientation discrimination," in *Proceedings of International Conference on Image Processing (ICIP '04)*, vol. 1, pp. 589–592, 2004.

[18] P. Wang and Q. Ji, "Learning discriminant features for multi-view face and eye detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, vol. 1, pp. 373–379, 2005.

[19] Y. Abramson and B. Steux, "YEF real-time object detection," in *Proceedings of the International Workshop on Automatic Learning and Real-Time (ALART '05)*, pp. 5–13, 2005.

[20] Y. Hori, K. Shimizu, Y. Nakamura, and T. Kuroda, "A real-time multi face detection technique using positive-negative lines-of-face template," in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR '04)*, vol. 1, pp. 765–768, 2004.

[21] S. Paschalakis and M. Bober, "A low cost FPGA system for high speed face detection and tracking," in *Proceedings of IEEE International Conference on Field-Programmable Technology (ICFPT '03)*, pp. 214–221, December 2003.

[22] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin, and W. Wolf, "Embedded hardware face detection," in *Proceedings of the 17th IEEE International Conference on VLSI Design (ICVLSI '04)*, vol. 17, pp. 133–138, 2004.

[23] M. Yang, Y. Wu, J. Crenshaw, B. Augustine, and R. Mareachen, "Face detection for automatic exposure control in handheld camera," in *Proceedings of the 4th IEEE International Conference on Computer Vision Systems (ICVS '06)*, p. 17, 2006.

[24] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

[25] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[26] R. E. Schapire, "The boosting approach to machine learning: an overview," in *Proceedings of the MSRI Workshop on Nonlinear Estimation and Classification*, pp. 149–172, Berkeley, Calif, USA, 2002.

[27] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International Journal of Computer Vision*, vol. 38, no. 1, pp. 15–33, 2000.

[28] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[29] I. DeMacq and L. Simar, "Hyper-rectangular space partitioning trees, a few insight," Tech. Rep., Universite Catholique de Louvain, Louvain, Belgium, 2002.

[30] J. Mitéran, J. Matas, E. Bourennane, M. Paindavoine, and J. Dubois, "Automatic hardware implementation tool for a discrete adaboost-based decision algorithm," *EURASIP Journal on Applied Signal Processing*, vol. 2005, no. 7, pp. 1035–1046, 2005.

[31] Y. Wei, X. Bing, and C. Chareonsak, "FPGA implementation of AdaBoost algorithm for detection of face biometrics," in *Proceedings of IEEE International Workshop on Biomedical Circuits and Systems*, pp. 17–20, December 2004.