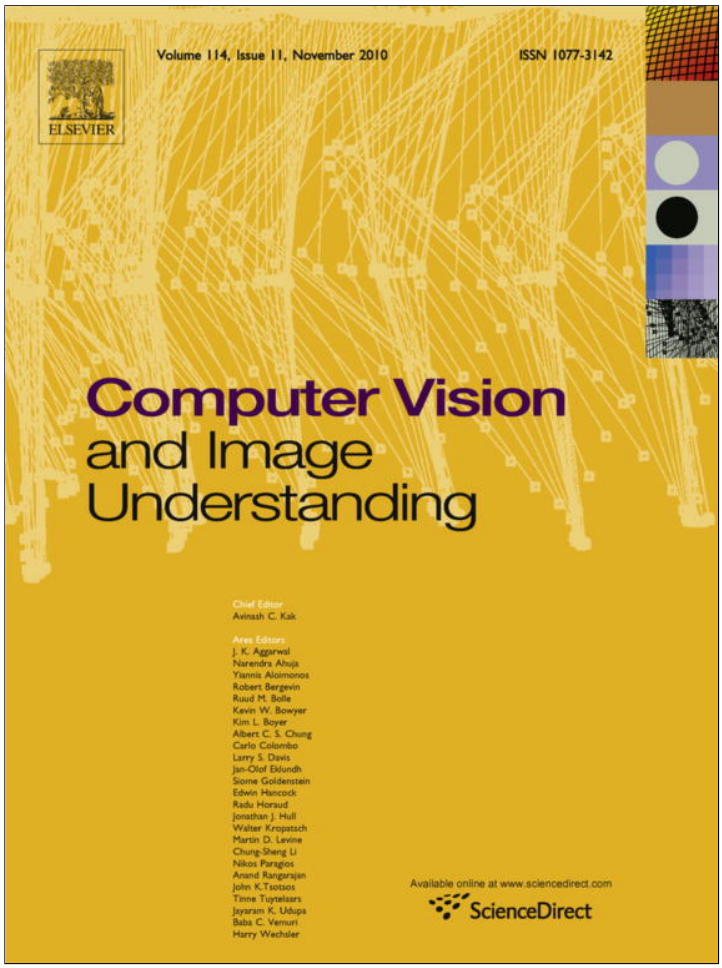


Computer Vision and Image Understanding 114 (2010) 1116–1125

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

AdaBoost-based face detection for embedded systems

Ming Yang [a,*], James Crenshaw [b], Bruce Augustine [c], Russell Mareachen [c], Ying Wu [d]

[a] NEC Laboratories America, Cupertino, CA 95014, United States
[b] Dolby Laboratories, Burbank, CA 91505, United States
[c] Motorola Inc., Schaumburg, IL 60196, United States
[d] EECS Dept., Northwestern Univ., Evanston, IL 60208, United States

ARTICLE INFO

Article history:
Received 9 January 2009
Accepted 17 March 2010
Available online 21 April 2010

Keywords:
Face detection
AdaBoost
FPGA
Genetic Algorithm

ABSTRACT

Face detection is a widely studied topic in computer vision, and recent advances in algorithms, low cost processing, and CMOS imagers make it practical for embedded consumer applications. As with graphics, the best cost-performance ratio is achieved with dedicated hardware. In this paper, we design an embedded face detection system for handheld digital cameras or camera phones. The challenges of face detection in embedded environments include an efficient pipeline design, bandwidth constraints set by low cost memory, a need to find parallelism, and how to utilize the available hardware resources efficiently. In addition, consumer applications require reliability which calls for a hard real-time approach to guarantee that processing deadlines are met. Specifically, the main contributions of the paper include: (1) incorporation of a Genetic Algorithm in the AdaBoost training to optimize the detection performance given the number of Haar features; (2) a complexity control scheme to meet hard real-time deadlines; (3) a hardware pipeline design for Haar-like feature calculation and a system design exploiting several levels of parallelism. The proposed architecture is verified by synthesis to Altera's low cost Cyclone II FPGA. Simulation results show the system can achieve about 75–80% detection rate for group portraits.

1. Introduction

Computer vision technology has experienced steady progresses both in theoretical study and practical applications in recent years. Many workable software and hardware systems have been proposed for video-based security surveillance [1,2], human–computer interaction (HCI) [3,4], intelligent traffic control [5], autonomous vehicles and robotic navigation [6], medical image processing [7], and machine vision in industrial control [8]. Besides conventional vision applications, popularity of handheld devices with cameras creates potential for new applications [9] in PDA's, cell phones [10], or any small battery driven devices. Meanwhile, the design methodologies [11] and computational capacity of embedded systems are soaring [9]. So, it is exciting from both a technical and commercial perspective to tailor algorithm development to the needs of low cost embedded vision systems.

For handheld cameras, human faces are a very common target of interest [12]. In addition, frontal face detection is usually the first step to initialize many computer vision tasks like tracking, recognition and image enhancement. In this paper we investigate the parallelism in the state-of-the-art AdaBoost-based face detection algorithm and port it to an embedded system for handheld cameras. The challenges mainly lie in efficient hardware architecture design, since most published vision algorithms do not take into consideration hardware characteristics and parallel processing which involves pipeline design, data flow arrangement, and parallel acceleration. For the algorithm, we improve the AdaBoost-based face detection on two aspects: in training stage, in order to fully exploit the given hardware resources, we propose to incorporate the Genetic Algorithm (GA) into the AdaBoost training to minimize the false positive rate given the number of Haar features; while in detection stage, for a hard real-time design we propose a new complexity control scheme in which unlikely candidate windows are skipped based on spatial correlation between successive scales.

We verify the cost of the hardware system by synthesizing for a low cost field programmable gate array (FPGA), suitable for integration in moderately-priced handheld cameras. Simulation results show this real-time detection system achieves 75–80% detection rate for group portraits.

The AdaBoost-based face detection algorithm and the Genetic Algorithm will be briefly reviewed in Section 2, as well as some other hardware face detection systems. In Section 3 we propose a new Genetic Algorithm based optimization for AdaBoost training and the hard real-time complexity control scheme. The system architecture design and complexity analysis are presented in

* Corresponding author. Fax: +1 408 863 6099.
E-mail addresses: myang@sv.nec-labs.com (M. Yang), james.crenshaw@dolby.com (J. Crenshaw), Bruce.Augustine@motorola.com (B. Augustine), Russell.Mareachen@motorola.com (R. Mareachen), yingwu@ece.northwestern.edu (Y. Wu).

1077-3142/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.cviu.2010.03.010

Section 4. The experimental results are given in Section 5. Concluding remarks are presented in Section 6.

2. Related work

2.1. AdaBoost-based face detection

The purpose of face detection is to locate any faces present in still images. This has long been a focus of computer vision research and has achieved great successes [13–16]. Comprehensive reviews are given by Yang [17] and Zhao [18].

Among face detection algorithms, the AdaBoost [19] based method proposed by Viola and Jones [20] has gained great popularity due to a high detection rate, low complexity, and solid theoretical basis. The fast speed of AdaBoost method is mainly due to the use of simple Haar-like features and a cascaded classifier structure, which excludes most of the image window hypotheses quickly.

In a pre-processing stage, an auxiliary image I_i, called the integral image or summed-area table [21] is calculated from the original image I_o, where the value $I_i(i,j)$ is the sum of pixels above or to the left of position (i,j) in I_o. Using I_i, the sum of pixel intensities in any rectangle in I_o can be calculated in constant time. Afterwards, each candidate image window w, at all positions and all scales, is fed into a cascaded classifier. At each stage, the classifier response $h(w)$ is the sum of a series of feature responses $h_j(w)$.

$$h(w) = \sum_{j=1}^{n_i} h_j(w), \quad h_j(w) = \begin{cases} \alpha_{j1} & f_j(w) < t_j \\ \alpha_{j2} & \text{otherwise} \end{cases}, \tag{1}$$

where $f_j(w)$ is the feature response of the jth Haar feature and α_{j1} and α_{j2} are the feature weight coefficients. If $h(w)$ is less than a threshold t, the candidate window w is regarded as non-face and thrown away, otherwise it proceeds to the next classifier. Multiple detections for one face are pruned by non-maximum suppression at the last step. Fig. 1 shows the block diagram. Please refer to Viola and Jones [16] and Lienhart [21] for the details.

Besides superior performance, the popularity of AdaBoost-based face detection also originates from low average execution time. As will be shown, it can be modified to allow for steady data flow and an efficient hardware implementation. However, there are challenges for an embedded system in that the algorithm assumes random access of the large integral image and considerable processing for multiplication and floating-point operations.

2.2. Genetic Algorithm optimization

Although the computational capabilities of chips are much more powerful than before and the transistors may be regarded free in general, given a specific hardware platform the resources including the computation units and storage are fixed and limited, which is one primary issue for hardware systems. It is always desirable to enhance the system performance given the available resources. For the AdaBoost-based face detection algorithm, the resources for integral image calculation and the imaging are determined by the image resolution, while the detection performance and the worst-case latency mainly depends on the number of Haar features used in the cascaded classifier as in Eq. (1). However, the optimization problem to minimize the expected number of Haar features N given a target false positive rate F and detection rate D is "tremendously difficult" [16], so in Viola and Jones's original algorithm they employed a greedy search scheme to select Haar features and build the classifier. In this paper we employ the Genetic Algorithm as an optimization tool to minimize the false positive rate given the number of Haar features while maintaining similar detection rate.

The Genetic Algorithm (GA) or Evolutionary Algorithm (EA) proposed by Holland [22] is an analog to creature population evolution, which can be viewed as a function optimizer [23]. In genetic algorithms, a sample point in the solution space of an optimization problem is often encoded as a binary string called "chromosome" or "genotype". A large population of "chromosomes" evolves according to the "fitness" or "reproductive ability" determined by a problem dependent evaluation function. During the evolution, mutation and crossover operations are performed on the population to generate new sample points in the solution space, which may be viewed as hyperplane sampling in the huge solution space. The typical usages of the Genetic Algorithm are concerned with nonlinear problems in which parameters cannot be treated as an independent variable and solved separately. Generally speaking, GA can be viewed as a global search method that does not rely on gradient information. A detailed discussion about GA is given in Whitley [23].

Recently, there have been some attempts to apply the GA in AdaBoost-based detection. In Treptow [24], evolutionary search was employed to select one feature in an enlarged Haar feature pool. The advantage is that instead of exhaustive search over all features evolutionary search can speed up the training and effectively find one good Haar feature in a quite large feature pool in reasonable time. Chen [25,26] used the GA to generate new positive training samples for AdaBoost face detector. Since the Haar features selected in the stage classifier are dependent on each other and there is no analytic relation between the number of features in classifier and its detection performance, this optimization problem to minimize the false alarm rate given the number of Haar features is nonlinear and hard, so well suitable for GA optimization. In addition, the classifiers which are a combination of Haar features can also be described as hyperplanes, as will be further explained in Section 3.

2.3. Hardware systems for face detection

In the literature, there have been some attempts on hardware implementations of diverse face detection algorithms where the

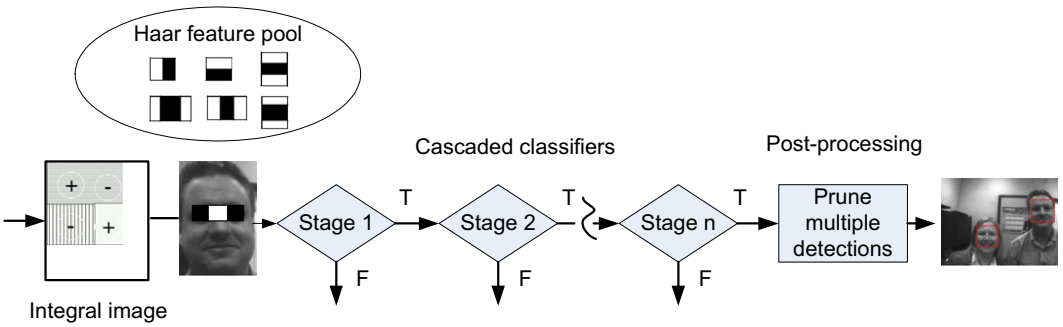


Fig. 1. Block diagram of AdaBoost-based face detection.

data flow is regular and parallelism can be easily identified, for example, tone color detection based methods [27,28] or Neural Networks based methods [29]. For skin-color methods, in the training stage a statistical skin-color model in a certain color space is learned with labeled skin pixels. During detection, skin pixels are extracted with this model, and then heuristics based on face edge templates [27] or connected component analysis [28] are applied to determine face regions. Generally, skin-color methods are efficient and robust to geometric transformation. Nevertheless, no matter what color space is used, skin-color models are not reliable in unconstrained environments since illumination conditions and variation among individuals cause considerable face color changes. Synthesis results were shown for the Neural Network method [29]. This Neural Network method is not computationally efficient. It took several hundred cycles to process one image window, therefore only 965 candidate windows were evaluated in each 300×300 image, which may considerably compromise the detection performance.

Recently, hardware implementations of the AdaBoost-based face detection algorithm were investigated in [30–33] with some preliminary results reported. A parallel architecture for AdaBoost algorithm was very briefly discussed in [31] and was synthesized using a commercial library targeted for a 500 MHz clock cycle. Bigdeli et al. [32] studied the effects of replacing certain software bottleneck operations by custom instructions on embedded processors, especially the image resizing and floating-point operations, but did not fully implement the entire algorithm in hardware. Lai et al. [33] presented a fairly complete design for network-on-chip (NoC) architecture that was verified on a Xilinx Virtex-II Pro FPGA, but no face detection performance was systematically evaluated. In this paper, we identify several practical issues arising from the hardware design and present a detailed investigation of reasonable solutions for detection algorithm design and hardware implementation.

3. Motivation and algorithm design

To port the AdaBoost face detection algorithm to an embedded system, tailoring the algorithm to efficiently and effectively utilize the hardware resources is critical. Given a specific hardware platform, the computational capability is fixed. Then, for an AdaBoost face detector the computational efficiency is mainly determined by the number of Haar features can be implemented on this hardware platform and the number of candidate image windows evaluated at run-time. Besides the computational capability, the bandwidth constraint between off-chip memory and processing engines are often the bottleneck for the entire system. Unlike a desktop implementation, which includes a large and expensive cache memory hierarchy, the random access of memory could be expensive for an embedded system. Therefore, finding of a regular data flow will greatly enhance the efficiency of RAM access and facilitate a pipeline arrangement.

The number of Haar features implemented on the hardware mainly determines the silicon area of the design and the worst-case latency. Given the available hardware resources on a specific platform, the maximal number of Haar features is bounded. The original feature selection method in [16] is a greedy approach, searching for the current best Haar feature in the feature pool and adding it into the classifier, then testing if the pre-defined requirements of detection rate and false positive rate are satisfied, if not it continues to search for the next best Haar feature. To improve the selection of Haar features in the classifier, we introduce the GA optimization method in the AdaBoost training stage.

In practice, in addition to the discrimination power of classifiers, the number of candidate image windows evaluated also plays a significant role in detection accuracy. The number of features examined at run-time is a data dependent variable. So, a complexity control scheme is indispensable to meet hard real-time deadlines of the hardware platform. We propose a complexity control method that exploits the spatio-temporal correlation between the image windows, to skip some unlikely image windows and increase detection rate when computational resources are overloaded.

In addition, to reduce hardware complexity and abide by real-time restrictions, we make some modifications to the AdaBoost-based algorithm. In the following, we will introduce the GA optimization method in the AdaBoost training stage, the hard real-time control scheme in the detection stage, and the algorithmic modifications, respectively.

3.1. Haar feature selection with the Genetic Algorithm

Generally speaking, appearances of faces can be regarded as a nonlinear manifold in a high dimensional space, e.g., a 400-dimensional space for 20×20 image windows. Each Haar feature with a threshold in Viola–Jones's detector can be regarded as a hyperplane in this space, hence one classifier with n_i Haar features splits the space into 2^{n_i} hypercells. One stage classifier describes a hyperobject that contains the face manifold, and the next classifier further cuts off non-face hyperregions from this hyperobject to delineate a more accurate approximation to the face manifold. The total number of features in the Haar feature pool is usually quite large, e.g., 43,844 in our training, so finding the optimal classifier with n_i features is a quite hard optimization problem. The GA is essentially a hyperplane sampling process in the huge solution space, which is a natural choice to solve this complicated optimization problem.

We employ the GA to further improve the AdaBoost classifier obtained by the greedy scheme. For a classifier with n_i features, we first generate the initial populations of S "chromosomes", where each chromosome is a classifier with n_i Haar features $\{f_{11}, \ldots, f_{1n_i}\}$. We will use the term chromosome and classifier alternately in this section. The evaluation function of the "fitness" of each chromosome is the false positive rate on the training set, since this is the variable we intend to optimize. Another choice is the expected error on the entire training set. Note if the detection rate is below a pre-defined value this "chromosome" loses the chance to survive, which guarantees the similar detection rate as the original training. After performing mutation and crossover operations with certain probability on each chromosome, the chromosomes whose fitness are below the average are replaced by the best chromosome, while the others survive to the next generation. In this simple GA scheme, the best chromosome in each generation always has the dominant populations and greater chance to further evolve into a better classifier with higher "fitness". The one with the lowest false positive rate in the last generation is returned as the final classifier. The mutation and crossover operations for one chromosome are defined as:

- **Mutation.** With the probability $P_m = 0.1$, one of the Haar features in the classifier is replaced by the best feature in the remaining feature pool.
- **Crossover.** With the probability $P_c = 0.5$, half of the Haar features in the classifier are exchanged with another randomly selected classifier.

The entire GA procedure is summarized as follows:

- **Initialization.** The initial set of chromosomes are generated by the greedy scheme in the original AdaBoost training.

- **One generation evolution.** For each generation, do the following operations to each chromosome:
  - **Mutation and crossover.** Perform mutation and crossover operations with certain probability to each chromosome.
  - **AdaBoost learning.** AdaBoost learning to determine the optimal threshold for each classifier.
  - **Chromosome evolution.** Evaluate the fitness of each chromosome and replace those whose fitness lies below the average by the best one.
- **Final classifier selection.** Select the best classifier from chromosomes at the last generation.

As shown in the experiments, as more generations are used in the GA optimization, the false positive rate drops from the order of 10^{-6} to 10^{-7}.

3.2. Hard real-time complexity control

There may be hundreds or even thousands of features in a detector. Although, according to [16], 80–90% of candidate image windows are skipped after the first 2 stage classifiers, the provable upper bound on number of Haar features evaluated at run-time for one frame is very high. For a hard real-time success, a complexity control mechanism is indispensable to guarantee that every frame can be processed in the exact designated time interval.

The straightforward solution is to truncate processing at given deadlines no matter how many image windows are processed. However, then faces present in latter candidate windows may be missed. Our scheme to control the run-time complexity meets hard real-time deadlines and has graceful degradation when the time budget is critical, at a cost that the detection rate may drop about 5%.

Given a clock frequency, a desired frame rate, throughput of classifier pipeline, and overhead of integral image calculation, we can estimate how many features can be evaluated in one frame, denoted as F_{frame}. For a specific image resolution and scaling factor, we know the maximal number W_{frame} of candidate image windows. The goal of complexity control is to make the best use of F_{frame} in case not all W_{frame} windows can be examined. Basically, we exploit the spatio-temporal correlation of image windows to make predictions to guide the classifiers and skip unlikely windows. This is based on the observation that if one image window passes all the tests of classifiers on a certain scale, it is very likely that the same region at the smaller scale passes more stage classifiers than average. An example is illustrated in Fig. 2 where the spatial distributions of the number of stage classifiers passed by the candidate image windows are drawn for the rightmost portrait image. In the 3 graphs on the left of Fig. 2, the X–Y axis indicates the spatial locations of candidate windows at three successive scales and the Z axis shows the number of stage classifiers that the candidate windows have passed. We can observe that around the true position of the face, i.e., the highest peaks of the three graphs on the left, the spatial distributions of the number of classifiers one candidate window pass are quite similar at different scales. Another observation is that for most video sequences, for portraits in particular, there is strong temporal correlation between frames.

Assume one window $w_{\{s_i\}}$ at scale s_i is rejected at the kth classifier, we denote it as $n(w_{\{s_i\}}) = k$ (if $w_{\{s_i\}}$ passes all N stages,

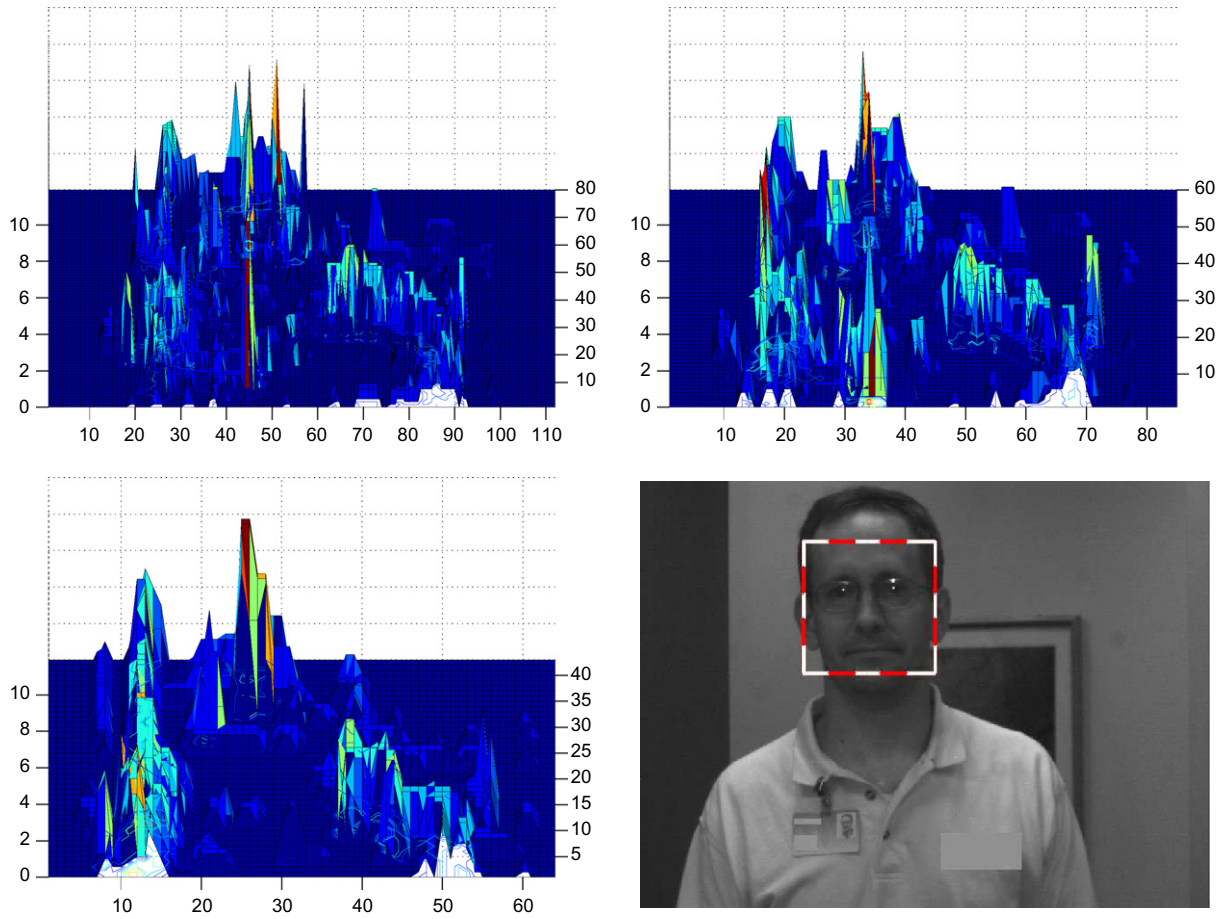


Fig. 2. The spatial distributions of the number of stage classifiers the candidate windows pass at three successive scales for a portrait image.

$n(w_{\{s_i\}}) = N + 1$). Our idea is to use the average of $n(w)$ at the smaller scales to guide the search at the larger scale. If we run out of the time budget at a frame and have to omit some scales, we intentionally make up these scales in the next frame. Specifically, for scale s_i an integral table, a summed-stage table (SST) of $n(w_{\{s_i\}})$ which records the accumulated sum of $n(w_{\{s_i\}})$, namely the number of stage classifiers at which candidate windows are being rejected, is built during the evaluation process. From the position of larger window $w_{\{s_{i+1}\}}$, we can determine the set of windows $O(w_{\{s_{i+1}\}})$ that have overlapped regions with $w_{\{s_{i+1}\}}$ at the scale s_i and calculate the average $\overline{n(w_{\{s_i\}})}$ in constant time with the SST which is just like calculating the Haar feature responses with the integral image I_i. If $\overline{n(w_{\{s_i\}})}$ is below a certain threshold T_n, this window is skipped without being tested by the rest of stage classifiers, and $n(w_{\{s_{i+1}\}})$ is set to T_n, which means it plays no role at the next scale. The entire procedure is summarized in Fig 3.

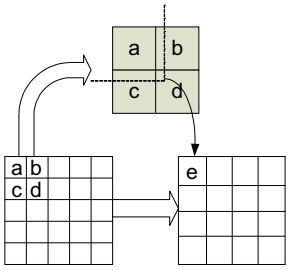
The detection rate may drop 5% if some face windows are skipped erroneously. However, all face candidate windows still pass all stage classifiers in the cascade, which implies that false positives do not increase. This is a desirable result, since usually applications prefer sacrifice of detection rate over more false positives to avoid showing regions which are incorrectly classified as faces. The experimental results of the complexity control scheme will be presented in Section 5.

3.3. Detection algorithm modifications

Some modifications and simplifications are applied to the algorithm to facilitate hardware implementation. Specifically, we re-scale the integral image on the fly; fixed-point numbers are used instead of floating-point ones for the coefficients and thresholds in the AdaBoost classifiers; the Haar feature pool is reduced; and we approximate the normalization factor for lighting condition correction.

Although [16] suggested that re-scaling a frame is more expensive than re-scaling the features, this is not always so. For an embedded system without a large cache, the loss of data locality incurred by the larger re-scaled features is worse than the cost of including a small re-scaling block, because loss of locality implies the main memory is accessed for each use of integral image data – beyond the bandwidth affordable in a low-cost device. But a re-scaler runs in parallel, takes little logic, and not much bandwidth. We re-sample a 25×25 block to a 20×20 and write back to the integral image store as shown in Fig. 4. Thus, only Haar features for 20×20 window need to be stored and applied, which greatly reduces the bandwidth since data brought in is re-used for adjacent windows. Another benefit of scaling factor 1.25 is that only shift and add operations are required.

Although simple Haar features enable fast lighting correction by maintaining another integral image of sum of squared pixels, this is expensive in terms of storage and bus traffic. Here we approximate the normalization factor σ of image window w by the average sum



e = a + (b-a)/4 + (c-a)/4 + [(d-a) – (b-a) – (c-a)]/16

Fig. 4. The re-scaling procedure for an integral image.

$\sigma = \frac{f(w)}{A(w)}$, where $f(w)$ is the sum of pixels in w and $A(w)$ is the area of w. With this approximation, every feature response only involves one multiplication, $f_i(w) < t_i \cdot f(w)$ instead of division.

Other modifications include the conversion from floating-point to fixed-point for thresholds, t_i, and coefficients, α_j. Specifically, we use 30 bits for t_i and 15 bits for α_j. For simplicity, only the six types of Haar features shown in Fig. 1 are implemented instead of the enhanced 45° tilted Haar feature set described in [21], which have no fundamental difference from the rectangular features. Empirically, these approximations work well (less than 1% loss in detection rate) for the majority of portrait images tested.

4. System design

Four instances of parallelism are considered. At task-level, a pipeline can be formed between acquiring frames, computing integral frames, and detecting faces within frames. Detecting faces at two different positions is parallelizable. Feature calculations can be overlapped. Lastly, feature calculation and image re-scaling can run in parallel.

4.1. System architecture

One goal for this design is to ensure the hard real-time deadline of providing a list of face locations once per image capture time. Another goal is suitability for use as a block in a chip. Fig. 5 shows the system block diagram, and illustrates how our design can be situated with a CPU, external memory, image sensor, and image sensor interface. It is assumed that the internal bus and memory interface provide bandwidth and latency guarantees to the Integral Image Computer and the Face Detection Engine (Fig. 6).

Top level tasks for face detection and a resolution of their data dependence is shown in Fig. 7. Two frames can be overlapped since there is no data dependence between them. Exposure time will be overlapped, so that the total time per frame, T_f, are $T_r + T_d$ milliseconds, where T_r is the readout time for the frame, and T_d is the time

Complexity Control Procedure	For all scale $\{s_0,\cdots,s_K\}$ and every window $w_{\{s_i\}}$
Parameters: The number of feature budget: F_{frame} The number of image windows: W_{frame}, $F_{average}=F_{frame}/W_{frame}$	1, If $C_F>F_{frame}$ goto 5 2, If $C_F/C_W<F_{average}$ or $s_i = s_0$ goto 3 Else Calculate $O(w_{si+1})$ and $\overline{n(s_i)}$ from $SST(s_i)$ If $\overline{n(s_i)} > T_n$ goto 3 Else $n(w_{s_i}) = T_n$ and goto 4 3, Evaluate $w_{\{s_i\}}$ and obtain $n(w_{s_i})$ 4, Update $SST(s_{i+1})$ 5, Record s_i and proceed to next frame.
Variables: Counter of evaluated features: C_F Counter of evaluated windows: C_W Summed-stage table: $SST(s_i)$ Overlapped window Set: $O(w_{si+1})$	

Fig. 3. The pseudocode for the complexity control procedure.

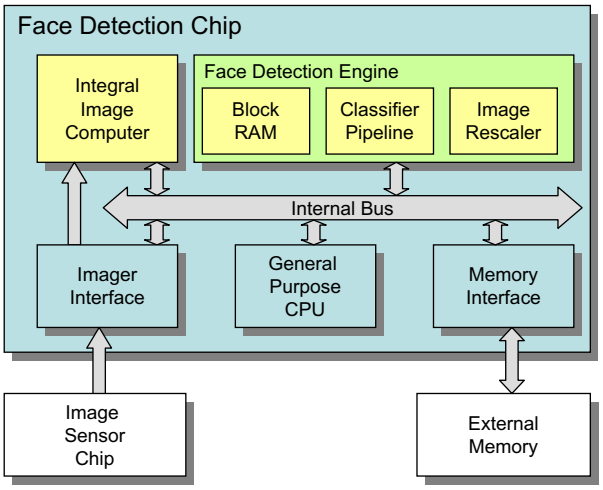


Fig. 5. The system architecture.

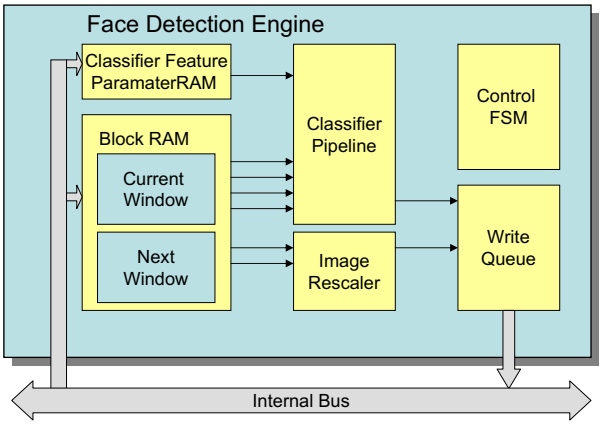


Fig. 6. The face detection engine.

during which detection is done. The exposure time, T_e can exceed T_f, but this case only makes detection deadlines easier.

Rather than moving image data out to main memory and then back in for integral image computation, a small hardware block is connected directly to the Imager Interface. This is shown in Fig. 5 as the Integral Image Computer. With two more line store RAMs, the block could also compute rotated integral images such as those in [21].

At the top level of the Feature Detection Engine, the tasks of fetching integral data from main memory, computing the classifiers on the current windows, re-scaling the integral image, and writing results to main memory run in parallel. Fig. 6 shows the top level design.

Use of a block RAM is illustrated in Fig. 8. Each external memory location is loaded twice: first as the bottom part of the block of windows, and then again in the top when the next block RAM row is brought in. As shown, the internal RAM has four read ports, which limits throughput to 1 feature per 2 cycles.

To understand the Classifier Pipeline design, it is necessary to have an idea of the procedure for computing a feature detection, which is shown as pseudocode in Fig. 9. The Classifier Pipeline design uses the integral image data to find weighted image areas for comparison. Overlapping execution of the pipeline for three successive feature calculations, A, B, and C, is shown in Fig. 10. Latency of the pipe is 5 cycles, but throughput is 1 feature per 2 cycles. The example shows the hardest feature type, requiring eight values from the integral image data. It is easy to modify the pipeline to find the rotated features in [21].

4.2. Performance analysis

Image sensor frame rate and pixel rate set the overall system deadlines and integral image processing times, respectively. However, these are not limiting factors. External memory bandwidth, B (in MB/s) and feature calculation throughput, C (in cycles per feature) are the critical values. Several second order effects are ignored, including latencies of feature response calculation and integral image computation, which are hidden by overlapped execution.

From B and C, two related values can be derived, which make the overall discussion clearer. The first is the rate at which blocks of integral image data are delivered to the detection engine. This is determined by the height of the block RAM in Fig. 8 which is 32 high, so each integral frame data value is loaded twice, for total external bandwidth of $2 \times 310K$ pixels/frame $\times$ 4 bytes/pixel $\times$ - frame rate fps, which for a frame rate of 10, would be 24 MB/s (note that the 4 bytes/pixel comes from the size used for the integral data).

The second derived value is cycles per integral block. For the example with block height 32, there are four 20 pixel-high windows per block, so cycles per block are $4 \times$ *window_rate* cycles/window. *Window_rate* is not constant, so here we use an average of 20 features per window. For the described feature pipeline, we have 40 cycles per window. The average is thus 160 cycles per block. Since the next block to the right overlaps by 16 out of 20 pixels, during the 80 cycles of computation we must bring in 4×32 integral frame values.

The minimum time spent on a block is thus governed by the rate at which blocks are brought in, so even if the classifier terminates the testing early in only one feature per window, the system still takes at least 128 cycles for the block. It would be nice to gain

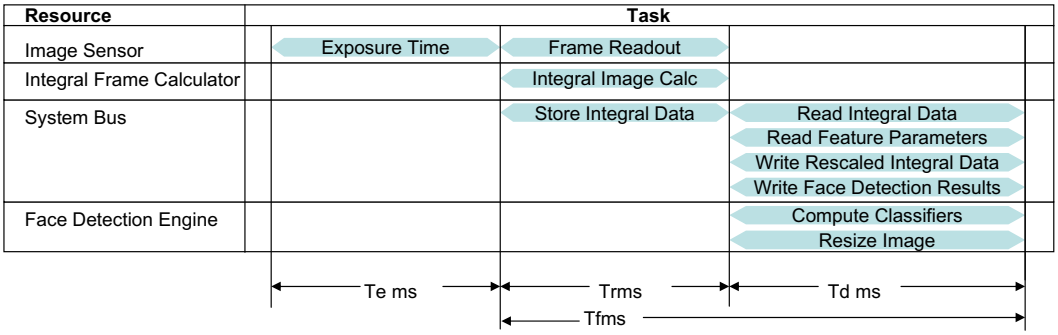


Fig. 7. Resource usage for each task and data dependence among tasks in a given frame. Note that exposure time will be overlapped with the previous frame's computation.

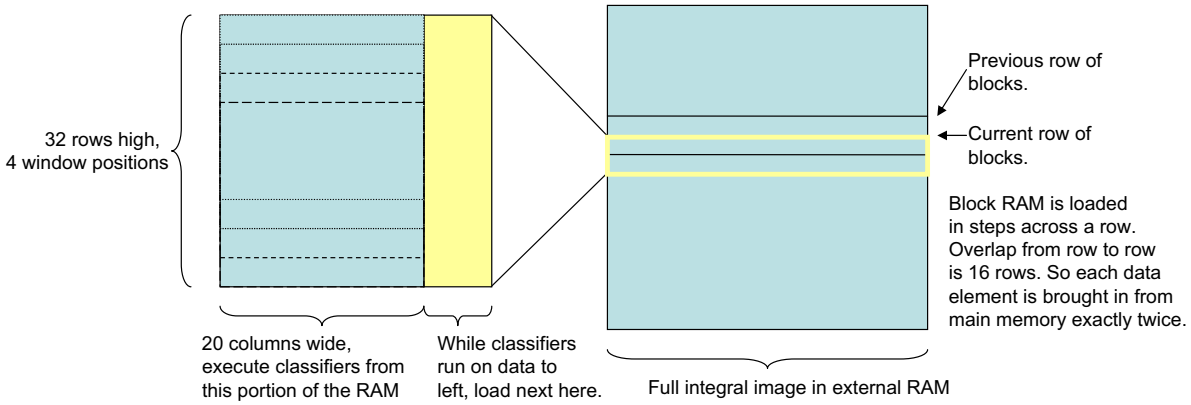


Fig. 8. Use of the Block RAM.

```
/* set RAM address lines */          6: RAM_port1_addr<-e
/* data is ready next cycle */       7: RAM_port2_addr<-f
1: RAM_port1_addr<-a                 8: RAM_port3_addr<-g
2: RAM_port2_addr<-b                 9: RAM_port4_addr<-h
3: RAM_port3_addr<-c                 /* step 10 finds sum of pixels *
4: RAM_port4_addr<-d                  * in rect 2 by integral image */
/* step 5 finds sum of pixels  *    10:Rect2_sum = RAM_port4_data-RAM_port3_data
 * in rect 1 by integral image */                  -RAM_port2_data+RAM_port1_data
5: Rect1_sum = RAM_port4_data        11:window_weight = window_sum
               -RAM_port3_data                       * feature_weight
               -RAM_port2_data       /* step 12 finds value to use for comparison */
               +RAM_port1_datav      12:decision = Rect2_sum - Rect1_sum - window_weight
```

Fig. 9. The pseudocode for feature calculation.

	cy1	cy2	cy3	cy4	cy5	cy6	cy7	cy8	cy9
Block RAM Port 1	A1	A6	B1	B6	C1	C6			
Block RAM Port 2	A2	A7	B2	B7	C2	C7			
Block RAM Port 3	A3	A8	B3	B8	C3	C8			
Block RAM Port 4	A4	A9	B4	B9	C4	C9			
Rectangle Sum		A5	A10	B5	B10	C5	C10		
Multiply Stage 1	A11		B13		C13				
Multiply Stage 2		A11		B14		C14			
Multiply Stage 3			A11		B15		C15		
Multiply Stage 4				A11		B16		C16	
Threshold compare					A12		B17		C17

Fig. 10. The feature calculation pipe.

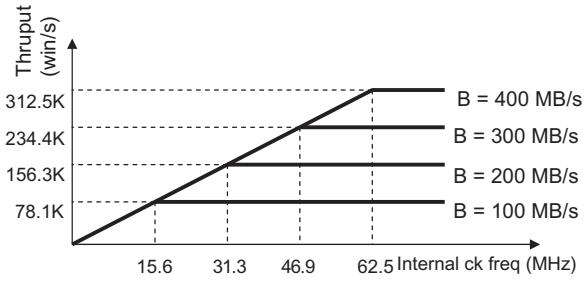


Fig. 11. Performance curves for different external bandwidth values.

back extra cycles, but it is not crucial since the rate per block is still better than average. The complexity control scheme covers this.

More generally, the maximum throughput of the system can be limited by the speed at which data can be delivered by main memory, or by the rate of computation in the feature detector (or both in actual operation due to data variation). Since feature detector throughput is a function of internal clock rate, and memory bandwidth depends on the number and speed of external RAM chips, a set of curves can be constructed showing system throughput as a function of internal clock rate for different external bandwidth values (see Fig. 11). In the graph, all values assume an average of 12 features per detection window. Adjusting the block RAM size alters these values.

5. Empirical results

5.1. Training settings

The training program is based on the implementation of OpenCV library [21]. The training data set employs 857 frontal faces and 3000 negative images. The minimal detection rate is 0.997 and the maximal false positive rate is 0.5 for each stage classifier. The AdaBoost detector trained for 20×20 windows includes 15 stage classifiers. The test set includes 133 upright frontal faces obtained from short portrait video chips and images on internet.

5.2. The GA optimization

We incorporate the GA optimization into the AdaBoost training and test the performance for different number of chromosomes and the total number of generations. Fig. 12 compares the false positive rate of the AdaBoost classifiers obtained by different training schemes, i.e., the original greedy search scheme and the GA optimization method. The dot line shows the false positive rate of the classifier using the original greedy search method implemented by [21]. The solid black line draws the false positive rate of each stage classifier before the GA optimization in our approach. The dash line illustrates the false positive rate of our approach using the GA optimization. The three figures from left to right in Fig. 12 show the performance using 5, 30, and 60 generations in

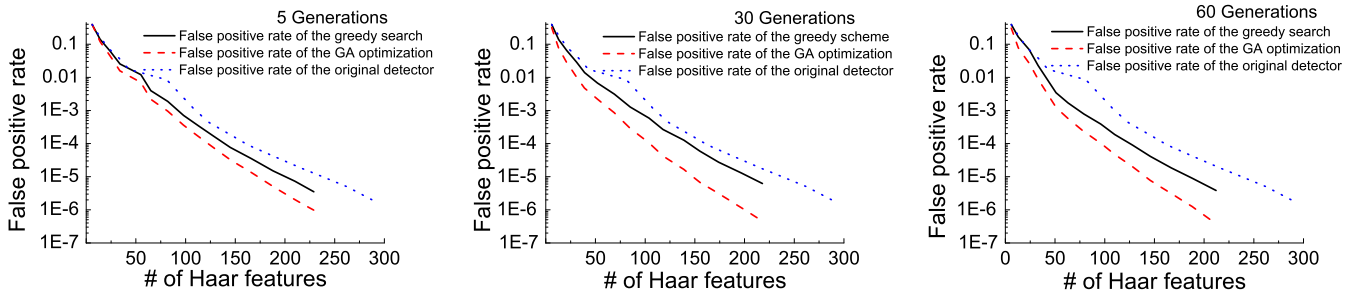


Fig. 12. Comparison of false positive rates.

the GA, where our approach lowers the false positive rate consistently. At the 15th stage classifier of the cascaded AdaBoost classifier, the false positive rate drops one order from 10^{-6} to 10^{-7} with the same number of Haar features. The detailed quantitative results of the false positive rate (FP) and the detection rate (DR) using different number of generations (# of gene.), number of chromosomes (# of chrom.), and number of selected features (# of feat.), are shown in Table 1. The improvement keeps increasing with the number of generations in the GA, which shows the effectiveness of our method.

The lower false positive rate of our approach is achieved at the cost of additional training efforts after obtaining the AdaBoost classifier using the greedy search scheme. The most computationally expensive module in training the AdaBoost classifier is to select the optimal Haar feature in the feature pool at each iteration, which involves testing the classification performance of each feature on the whole training data set and finding the threshold. Thus, the training time of the greedy search scheme is approximately proportional to the number of Haar features in one stage classifier, in contrast the training time of the GA method can be estimated from the number of generations since a stage classifier evolves as a chromosome. If 60 generations are used in the GA, the training time is about 60/212 = 28.3% more than that of the original greedy search.

5.3. Hard real-time control

In our proof-of-concept implementation, we use an FPGA chip with 50 MHz system clock. The overhead of integral image calculation and image capture is approximately 1 M cycles per frame. At 10 fps, 4 M cycles per frame are available, which yields $F_{frame} = 2M$. The total number of windows, $W_{frame} = 320K$, so the average budget is $\overline{F} = 6.25$, which is only slightly larger than the number of features in the first classifier. Without any control scheme, the actual number of features evaluated in run-time ranges from $3.5M$ to $4M$, which amounts to 10–12 features evaluated per window.

We compared our complexity control approach with the truncation scheme (where the detector moves on to the next frame if the hard real-time deadline is reached) in terms of the number of detected faces (# of DF), the number of false positive (# of FP), the detection rate (DR), and the average ratio (skipped ratio) of windows skipped to W_{frame} in our method. The detection result when no deadline is enforced is listed to demonstrate the detector's performance. The detection results are shown in Table 2. Some representative detection results on portraits with diverse face sizes are shown in Fig. 13.

Since the time budget to calculate Haar responses is only slightly larger than one half of what is needed, the truncation scheme skips the majority of windows on large scales. Thus the detection performance deteriorates dramatically in that scheme. In our scheme, the threshold T_n is 3, which is fairly conservative. On average 22% of the total windows are skipped based on the spatial correlations in the hard real-time control scheme in Section 3.2. If some large windows are omitted due to the deadline, they can be made up in the next frame. Our complexity control scheme slightly decreases the detection rate and false positive rate simultaneously by allocating more computational resources to more likely windows.

5.4. Synthesis results

To estimate the cost of the design, it was coded in Verilog and synthesized for the Altera Cyclone II FPGA family. Cyclone is low-priced with less capability than the Stratix line or Xilinx's Virtex line. Cyclones are good because it is feasible to use them directly in moderate volume applications, and designs which fit this family are very small in ASICs.

The particular design choices include a 120×24 Feature Engine Block RAM. This gives system bus headroom for transfers ignored in the earlier discussion. The Integral Computer also includes an output FIFO of about 30 Kbits, which was expedient but could be eliminated with further work (so it is not listed in Table 3). The Feature Engine throughput is 2 cycles per feature, based on 4 data ports as described earlier.

Table 3 shows the total use of logic. Logic Element count is based on complex programmable logic blocks found in FPGA chips, and using a rough rule of thumb, the design has 32 to 45 KGates logic (nand2 equivalent) and size is dominated by RAM.

The Memory Blocks column refers to Altera's 4 Kbit RAM blocks, so the design uses parts of 95 such blocks for a total of 22 KBytes of storage. Total size of the design is quite small, despite the lack of several area optimizations omitted due to design time.

5.5. Discussions

The overall detection performance without time constraint is lower than the results reported in [16] since we have made some

Table 1
Comparison of the greedy search (GS) and the GA optimization.

# of gene.	# of chrom.	# of feat.	FP of GS	FP of GA	DR of GA
5	10	229	3.51×10^{-6}	9.77×10^{-7}	0.97780
30	50	218	6.25×10^{-6}	4.55×10^{-7}	0.97199
60	80	212	3.83×10^{-6}	3.79×10^{-7}	0.97666

Table 2
Comparison of our approach and the truncation scheme.

Method	Total # of faces	# of DF	# of FP	DR	Skipped ratio
No deadline	133	107	15	80.5%	0
Truncation	133	68	10	51.1%	0
Our method	133	101	13	75.9%	22%

Fig. 13. Representative detection results.

Table 3
Synthesis results for primary modules.

	Logic elements	Memory bits	Memory blocks
Feature Engine Datapath and Control	1582	0	0
Feature Engine Data RAMs	45	31,284	10
Feature Engine Block RAM	4173	103,680	64
Integral Computer Datapath and Control	524	0	0
Integral Computer RAMs	70	34,633	21
Total	6394	197,226	95

tradeoffs between performance and implementation complexity. For example, the modification of lighting correction factor avoids maintaining another integral image of sum of squared pixels which generally requires 64-bit floating-point numbers to store. This modification significantly saves storage and bus bandwidth at the cost of lower detection rates at some extreme lighting conditions. Other tradeoffs such as using thresholds with low precision in the boosting classifiers and excluding the tilted Haar feature set have no major impacts on the performance. Given limited memory storage and bandwidth constraints, our approach is practical and suited for hardware implementation of the AdaBoost face detection algorithm. The Haar features are inherently insensitive to random image noise.

Handheld embedded architectures are rapidly evolving. We expect vision specific capabilities will be added in a manner similar to the evolution that graphics processing has gone through. In this paper, we have described a target FPGA architecture suitable for inclusion in a dedicated vision processing unit similar to the image processing chips found in camera modules or those included with the camera interface of applications processors. The advantages of doing an FPGA version include rapid prototyping, early timing closure, and having a way to quickly iterate and test design improvements. Furthermore, as a case study example, the complexity control scheme for enforcing the real-time execution, the buffering scheme utilizing the processor data cache hierarchy, and the parallelism and pipeline designs for the AdaBoost face detection proposed in this paper are also beneficial to other hardware platforms, e.g., DSP or OMAP.

6. Conclusions and future work

In this paper, we proposed an efficient embedded system design for AdaBoost-based face detection algorithm which exploits available parallelism. We identified some practical issues arising from the hardware design and presented a detailed investigation of reasonable solutions. The proposed GA based optimization of classifiers and the complexity control scheme are beneficial to any hard real-time implementation, whether hardware or software based. The proof-of-concept design can be synthesized for an FPGA costing as little as $20, which supports wide applicability for many consumer applications. There is still room to further improve the detection performance, so our future work includes increasing the throughput of detection engine pipeline to evaluate more image windows.

Acknowledgments

This work was supported in part by National Science Foundation Grant IIS-0347877 and IIS-0916607 and Motorola UPR Fellowship.

References

[1] W. Hu, T. Tan, L. Wang, S. Maybank, A survey on visual surveillance of object motion and behaviors, IEEE Trans. Syst., Man, Cybern. C 34 (3) (2004) 334–352.
[2] T.B. Moeslund, A. Hilton, V. Kruger, A survey of advances in vision-based human motion capture and analysis, Comput. Vis. Image Understanding 104 (2-3) (2006) 90–126.
[3] V.I. Pavlovic, R. Sharma, T.S. Huang, Visual interpretation of hand gestures for human–computer interaction: a review, IEEE Trans. Pattern Anal. Machine Intell. 19 (7) (1997) 677–695.
[4] A. Jaimes, N. Sebe, Multimodal human computer interaction: a survey, Comput. Vis. Image Understanding 108 (1-2) (2007) 116–134.

[5] Z. Sun, G. Bebis, R. Miller, On-road vehicle detection: a review, IEEE Trans. Pattern Anal. Machine Intell. 28 (5) (2006) 694–711.
[6] D.G.C. Race. <http://www.darpa.mil/grandchallenge>.
[7] I.N. Bankman, Handbook of Medical Image Processing and Analysis, Elsevier Inc., 2009.
[8] E.N. Malamas, E.G.M. Petrakis, M. Zervakis, L. Petit, J.-D. Legat, A survey on industrial vision systems, applications and tools, Image Vis. Comput. 21 (2) (2003) 171–188.
[9] B. Kisacanin, S.S. Bhattacharyya, S. Chai, Embedded Computer Vision, Springer Inc., London, 2008.
[10] X. Tang, Z. Ou, T. Su, P. Zhao, Cascade AdaBoost classifiers with stage features optimization for cellular phone embedded face detection system, in: International Conference on Advances in Natural Computation (ICNC), vol. 3, Changsha, China, 2005, pp. 688–697.
[11] M. Sen, I. Corretjer, F. Haim, S. Saha, J. Schlessman, S.S. Bhattacharyya, W. Wolf, Computer vision on FPGAs: design methodology and its application to gesture recognition, in: Workshop in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Diego, CA, 2005.
[12] H. Nozaki, Y. Motoki, H. Hibino, T. Maeda, T. Ohta, Digital camera, US Patent Application Publication 0,088,538, 2005.
[13] H.A. Rowley, S. Baluja, T. Kanade, Neural network-based face detection, IEEE Trans. Pattern Anal. Machine Intell. 20 (1) (1998) 23–38.
[14] K.-K. Sung, T. Poggio, Example-based learning for view-based human face detection, IEEE Trans. Pattern Anal. Machine Intell. 20 (1) (1998) 39–51.
[15] H. Schneiderman, T. Kanade, Object detection using the statistic of parts, Int. J. Comput. Vis. (IJCV) 56 (3) (2004) 151–177.
[16] P. Viola, M.J. Jones, Robust real-time object detection, Int. J. Comput. Vis. 57 (2) (2004) 137–154.
[17] M.-H. Yang, D. Kriegman, N. Ahuja, Detecting faces in images: a survey, IEEE Trans. Pattern Anal. Machine Intell. 24 (1) (2002) 34–58.
[18] W. Zhao, R. Chellappa, P. Phillips, Face recognition: a literature survey, ACM Comput. Surveys 35 (4) (2003) 399–458.
[19] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, in: European Conference on Computational Learning Theory, 1995, pp. 23–27.
[20] P. Viola, M.J. Jones, Rapid object detection using a boosted cascade of simple features, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, Hawaii, 2001, pp. 511–518.
[21] R. Lienhart, J. Maydt, An extended set of Haar-like features for rapid object detection, in: IEEE Conference on Image Processing (ICIP), vol. 1, New York, 2002, pp. 900–903.
[22] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.
[23] D. Whitley, A genetic algorithm tutorial, Stat. Comput. 4 (1994) 65–85.
[24] A. Treptow, A. Zell, Combining AdaBoost learning and evolutionary search to select features for real-time object detection, in: IEEE Congress on Evolutionary Computation (CEC), vol. 2, Portland, OR, 2004, pp. 2107–2113.
[25] J. Chen, X. Chen, W. Gao, Expand training set for face detection by ga re-sampling, in: IEEE Conference on Automatic Face and Gesture Recognition (FGR), Seoul, Korea, 2004, pp. 73–78.
[26] J. Chen, X. Chen, W. Gao, Resampling for face detection by self-adaptive genetic algorithm, in: International Conference on Pattern Recognition (ICPR), vol. 3, Cambridge, UK, 2004, pp. 822–825.
[27] Y. Hori, K. Shimizu, Y. Nakamura, T. Kuroda, A real-time multi face detection technique using positive–negative lines-of-face template, in: International Conference on Pattern Recognition (ICPR), vol. 1, Cambridge, UK, 2004, pp. 765–768.
[28] S. Paschalakis, M. Bober, A low cost FPGA system for high speed face detection and tracking, in: IEEE Conference on Field-Programmable Technology (ICFPT), 2003, pp. 214–221.
[29] T. Theocharides, G. Link, N. Vijaykrishnan, M. Irwin, W. Wolf, Embedded hardware face detection, in: IEEE Conference on VLSI Design (ICVLSID), Mumbai, India, 2004, pp. 133–138.
[30] M. Yang, J. Crenshaw, B. Augustine, R. Mareachen, Y. Wu, Face detection for automatic exposure control in handheld camera, in: IEEE Conference on Computer Vision System (ICVS), New York City, NY, 2006, p. 17.
[31] T. Theocharides, N. Vijaykrishnan, M. Irwin, A parallel architecture for hardware face detection, in: IEEE Symposium on Emerging VLSI Technologies and Architectures (ISVLSI), Karlsruhe, Germany, 2006, pp. 452–453.
[32] A. Bigdeli, C. Sim, M. Biglari-Abhari, B.C. Lovell, Face detection on embedded systems, in: International Conference on Embedded Software and Systems (ICESS), Daegu, Korea, 2007, pp. 295–308.
[33] H.-C. Lai, R. Marculescu, M. Savvides, T. Chen, Communication-aware face detection using noc architecture, in: IEEE Conference on Computer Vision System (ICVS), Santorini, Greece, 2008, pp. 181–189.