Face detection for automatic exposure control in handheld camera

Ming Yang[2] James Crenshaw[1]
Bruce Augustine[1] Russell Mareachen[1] Ying Wu[2]

[1]Motorola Labs, Schaumburg, IL 60196
{J.Crenshaw,Bruce.Augustine,Russell.Mareachen}@motorola.com
[2]EECS Department, Northwestern University, Evanston, IL 60201
{mya671,yingwu}@ece.northwestern.edu

Abstract

Face detection is a widely studied topic in computer vision, and 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. The challenges of face detection in embedded environments include bandwidth constraints set by low cost memory and a need to find parallelism. Consumer applications need reliability, calling for a hard real-time approach to guarantee that deadlines are met. We present a face detection system for automatic exposure control in a handheld digital camera or camera phone. Contributions include a complexity control scheme to meet hard real-time deadlines, 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 algorithm can achieve about 80% detection rate for group portrait pictures.

1 Introduction

Recent years have witnessed the steady progress of both theoretical study and practical applications in computer vision. Many workable software and hardware systems have been proposed for surveillance, robotic, human-computer interaction (HCI) [14], and intelligent traffic measurement [4]. Besides conventional vision applications, popularity of handheld devices with cameras creates potential for fantastic new applications in PDA's, cell phones, or any small battery driven device. Meanwhile, the design methodologies [9] and computational capacities of embedded systems are soaring. 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 [6]. In addition, frontal face detection is usually the first step to initialize many computer vision tasks like tracking, recognition and image analysis. In this paper we investigate the parallelism in the state-of-the-art Adaboost based face detection algorithm and use it in an embedded system for automatic exposure control. The challenges include efficient system design, since most published vision algorithms don't take hard real-time and parallel processing into consideration. The latter involves pipeline design, data flow arrangement and parallel acceleration, while for a hard real-time design, we propose a new complexity control scheme in which unlikely 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 some related hardware face detection systems will be briefly reviewed in Sec.2. In Sec.3 we propose the new hard real-time complexity control scheme. The system architecture design and complexity analysis are presented in Sec.4. The experimental results are given in Sec.5. Concluding remarks are presented in Sec.6.

2 Related works

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 seen great success [1, 8, 12] recently. There is too much literature on this topic to men-

Proceedings of the Fourth IEEE International Conference on Computer Vision Systems (ICVS 2006)
0-7695-2506-7/06 $20.00 © 2006 IEEE

tion here. Please refer to [13] and [15] for detailed surveys.

Among face detection algorithms, the Adaboost [2] based method proposed by Viola and Jones [11] gains great popularity due to a high detection rate, low complexity and solid theoretical basis. The fast speed of Adaboost method is 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 [5] is calculated from the original image, I_o, where the value $I_i(i,j)$ is the sum of I_o pixels above or to the left of position (i,j) in I_o. Using I_i, the sum of I_o pixel intensities in any rectangle can be calculated in constant time. Afterwards, each 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 features responses $h_j(w)$.

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

where $f_j(w)$ is the feature response of the jth Haar feature and $\alpha_{j1}\alpha_{j2}$ are the feature weight coefficients. If $h(w)$ is less than a threshold t, the window w will be regarded as non-face and thrown away, otherwise proceed to the next classifier. Multiple detections for one face will be pruned at the last step. Fig. 1 shows the block diagram. Please refer to [12] and [5] for the details.

Beyond performance, Adaboost face detector popularity comes from low average execution time. As will be shown, it can be modified to allow for steady data flow and a neat hardware implementation. However, there are challenges for an embedded system. The algorithm assumes random access of the large integral image and considerable processing power for multiplication and floating-point.

2.2 Hardware systems for face detection

In the literature, there are hardware implementations of face detection based on tone color detection [3, 7] and Neural Networks [10]. 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 template [3] or connected component analysis [7] are applied to determine face regions. Generally, skin-color methods are efficient and robust to geometric transformation. But, no matter the color space used, skin-color models are not reliable in unconstrained environments since illumination and variation among individuals cause face color changes. Synthesis results were shown for the Neural Network method [10]. However, the Neural Network is not computationally efficient, since it took several hundred cycles to process one window, and therefore only 965 windows were evaluated in each 300×300 image, which can reduce the detection rate.

3 Challenges and algorithm design

Essentially, the face detection problem is a pattern classification problem. In addition to the discrimination power of classifiers, the number of image windows evaluated plays a significant role in performance, so computational efficiency is critical to the success of detection. The number of features examined at run-time is an input image-dependent variable. So, a complexity control scheme is indispensable to meet hard real-time deadlines. Here we propose a complexity control method that exploits the spatiotemporal 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 approximations to the Adaboost based algorithm.

3.1 Hard real-time

There may be hundreds or even thousands of features in a detector. Although, according to [12], $80\% - 90\%$ of windows will be skipped after the first 2 stages, the provable upper bound on number of features evaluated for one frame is very very high. For hard real-time success, a complexity control mechanism guarantees that every frame can be processed in the exact designated time interval. A classic example of the need for this is the comparison of watching a DVD on a dedicated player versus watching it on a PC. Unlike the PC, which has no hard real-time guarantee from its OS, a dedicated player will not skip frames unless there is an external problem like a scratch on the disk.

The straightforward solution is to truncate processing at given deadlines no matter how many image windows are processed. However, this is not elegant and faces occurring in latter windows may be missed (if the image is searched from the finest scale to coarsest, then larger faces will be skipped). Here we propose a scheme to control the run-time complexity to meet hard real-time deadlines and achieve graceful degradation when the time budget is critical, while at the cost that the detection rate may drop a little bit.

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 scale factor, we know the maximal number W_{frame} of 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

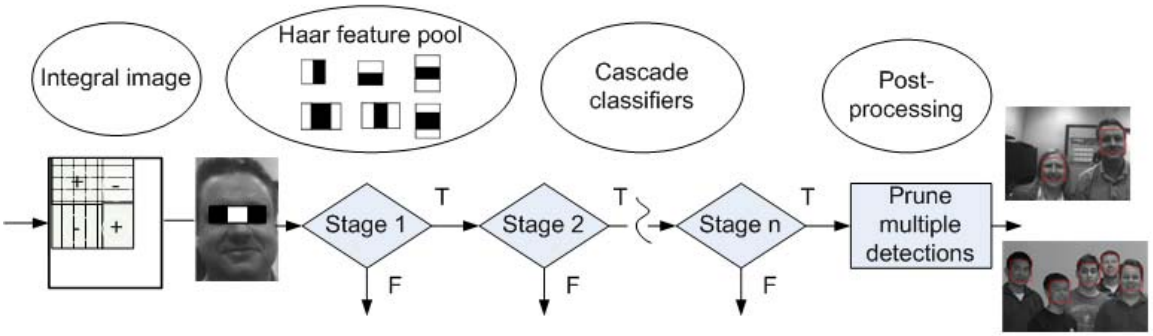

Figure 1. Block diagram of Adaboost based face detection.

the spatiotemporal correlation of images windows to make predictions to guide the classifiers and skip unlikely windows. This is based on the fact that if one image window passes all the tests of classifiers on a certain scale, there is a high probability that the same region at the smaller scale will pass more stages than average. The number of classifiers the windows pass for 3 successive scales are drawn in Fig. 2, which exhibits fairly strong similarity. Another fact is that for most sequences, and for portraiture 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, $n(w_{\{s_i\}}) = N + 1$). Our idea is to use the average of $n(w)$ on the smaller scales to guide the search on the larger scale. If we run out of all time budget and have to omit some scales, we intentionally make up these scales in the next frame. Specifically, for scale s_i an integral table, summed stage table (SST) of $n(w_{\{s_i\}})$ is built during the evaluation process. From the position of larger window $w_{\{s_{i+1}\}}$, we can determine the set of windows $O(w_{\{s_{i+1}\}})$ who 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. If $\overline{n(w_{\{s_i\}})}$ is below a certain threshold T_n, this window will be skipped, and $n(w_{\{s_{i+1}\}})$ is set to T_n, which means it will play no role in the next scale. The entire procedure is summarized in Fig 3.

The detection rate may drop a little if some face windows are skipped erroneously. However, all face candidates still pass all classifiers in the cascade, which implies that false positives won't increase. This is a desirable result, since our application of automatic exposure control prefers sacrifice of detection rate over more false positives to avoid setting exposure based on regions which were incorrectly classified as facial. The experimental results of the complexity control scheme will be presented in Sec.5.

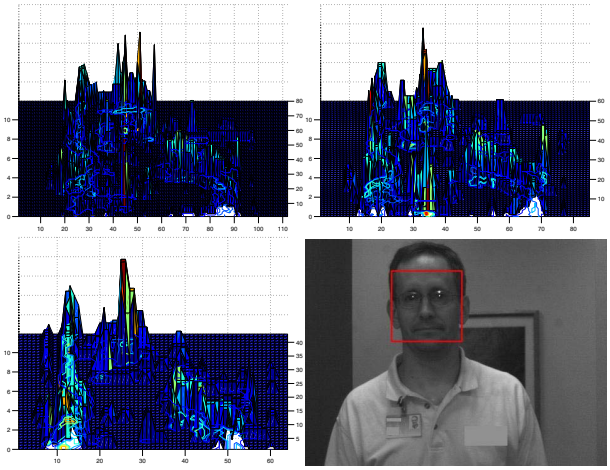

Figure 2. The number of classifiers the windows pass for 3 successive scales.

3.2 Detection algorithm approximations

Some modifications and simplifications were applied to the algorithm to accelerate hardware implementation. Specifically, we re-scale the integral image on the fly; fixed-point is used instead of floating-point; the Haar feature pool is reduced; and we approximate the normalization factor.

Although [12] 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 will be 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 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,

Complexity Control Procedure	For all scale $\{s_0, \cdots, s_K\}$ and every window $w_{(s)}$
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 $n(w_s) = T_n$ and goto 4 3, Evaluate $w_{(s)}$ and obtain $n(w_s)$ 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: $O(w_{si+1})$	

Figure 3. Pseudocode for complexity control procedure.

which greatly reduces the bandwidth since data brought in is re-used for adjacent windows. Another benefit of scale factor 1.25 is that only shift and add operations are required.

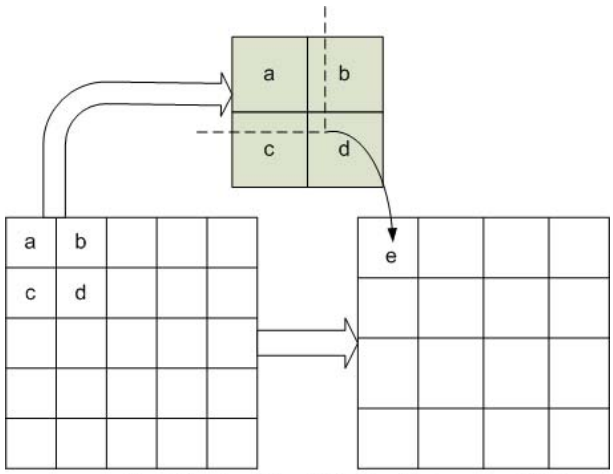

Figure 4. Re-scaling procedure for integral image.

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 $\sigma = \frac{f(w)}{A(w)}$, 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, $f_i(w) < t_i \cdot f(w)$.

Other modifications include the conversion from float to fixed-point for thresholds, t_i, and coefficients, α_j. Specifically, we use 30 bits for t_i and 15 for α_j. For simplicity, only the 6 features shown in Fig. 1 are implemented instead of the enhanced 45^o tilted Haar feature set described in [5], which have no fundamental difference from the rectangular features. These approximation work well for the majority of images.

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 (Fig. 7). 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.

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, Tf, will be Tr + Td milliseconds, where Tr is the readout time for the frame, and Td is the time during which detection is done. The exposure time, Te can exceed Tf, 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.

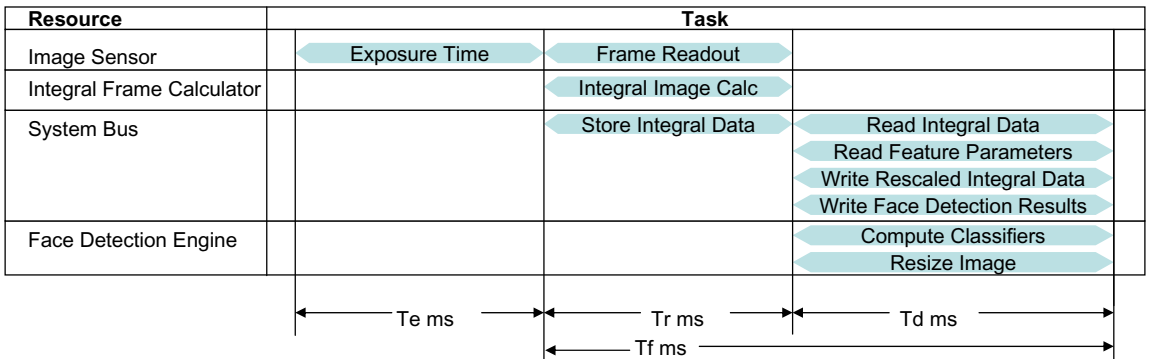

Figure 7. Resource usage for each task and data dependence among tasks in a given frame. Note that exposure time will be overlapped with the previous frame's computation.

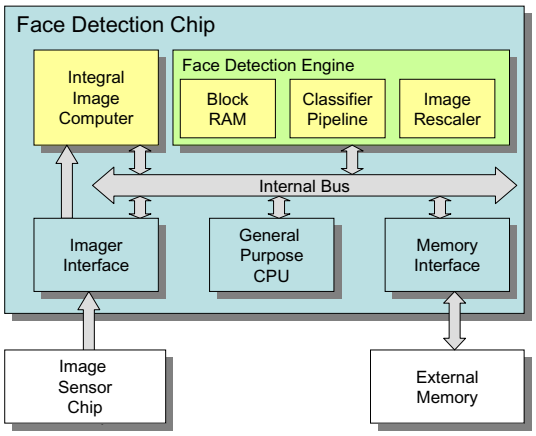

Figure 5. System Architecture.

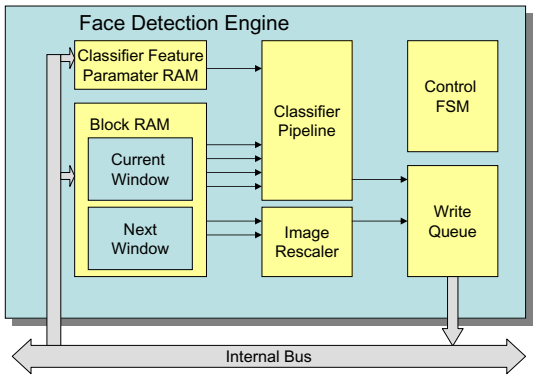

Figure 6. Face Detection Engine.

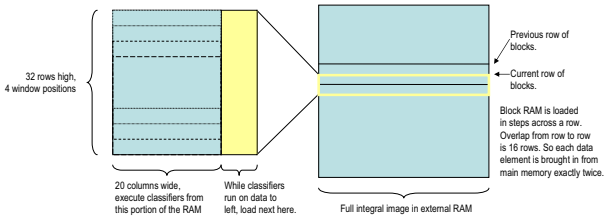

Figure 8. Use of the Block RAM.

With two more line store RAMs, the block could also compute rotated integral images such as those in [5].

At the Feature Detection Engine top level, 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.

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 4 read ports, which limits throughput to 1 feature per 2 cycles.

The Classifier Pipeline design uses the Integral Frame 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. 9. 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 [5].

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. External memory bandwidth, B in MBytes per second and 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

Figure 9. Feature calculation pipeline. Three successive feature calculations.

throughput, C in cycles per feature are the critical values. Several second order effects are ignored, including latencies of feature response calculation and integral frame 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. Fig. 8 is 32 high, so each integral frame data value is loaded twice, for total external bandwidth of $2 \times 310K$ pixels/frame $\times$ 4bytes/pixel $\times$ frame rate fps, which for a frame rate of 10, would be 24M bytes per second (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 (a point addressed by the complexity control scheme), 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 exits in only one feature per window, the system will take at least 128 cycles for the block. It would be nice to gain back extra cycles, but it's 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. 10. In the graph, all values assume an average of 12 features per detection window. Adjusting the block RAM size alters these values.

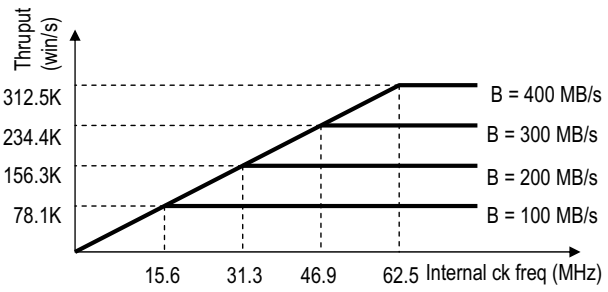

Figure 10. Performance curves for different external bandwidth values.

Analysis of the design size shows that it fits on a low cost FPGA and would be a very small block on an ASIC. Size is dominated by the block RAM, but also includes multiplexing between block RAM and arithmetic units as well as line stores for computing the integral frame. The Empirical results section confirms that the design is small.

5 Empirical results

The training employs 858 frontal faces and 3000 negative images. The minimum hit rate is 0.999 and the maximal false positive rate is 0.4 for each stage classifier. The Adaboost detector trained for 20×20 windows includes 11 stages and 140 features total. The test set includes 133 upright frontal faces obtained from short portrait video clips and images on internet.

In our proof-of-concept implementation, we use an FPGA chip with 50MHz system clock. The overhead of integral image calculation and image capture is approximately 1M cycles per frame. At 10fps, 4M 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 to 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

Table 1. Comparison of our approach and the truncation scheme.

Method	Total # of faces	# of DF	# of FP	DR	skipped ratio
OpenCV	133	124	3	93.2%	0
No deadline	133	107	15	80.5%	0
Truncation	133	68	10	51.1%	0
Our method	133	101	13	75.9%	22%

method. The detection result when no deadline is enforced is listed to demonstrate the detector's performance. The result of OpenCV's [5] implementation is listed for comparison. Note that since there are 25 stages and 2913 Haar features in total, on average about 50 Haar features are evaluated for each frame, which means far more resources are required than our 11-stage detector. This result shows the difficulty of our test set. The detection results are shown in Table 1. Some representative detection results are shown in Fig. 11.

Since in our test case the feature budget is only slightly larger than one half of what is needed, truncation 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. 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. The detection performance without time deadline is lower than the results of OpenCV's implementation, which is partly due to insufficient training efforts and the approximation of lighting correction factor.

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 a largish output FIFO, which was expedient but could be eliminated with further work. The Feature Engine throughput is 2 cycles per feature, based on 4 data ports as described earlier.

Table 2 shows low 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.

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 complexity control scheme is 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. To further improve the detection performance, our future work includes optimizing the Adaboost training given the number of features, and increasing the throughput of detection engine pipeline to evaluate more image windows.

References

[1] H. A.Rowley, S. Baluja, and T. Kanande. Neural network-based face detection. *IEEE Trans. Pattern Anal. Machine Intell.*, 20(1):23–38, Jan. 1998.

[2] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory (EuroCOLT)*, pages 23–27, 1995.

[3] 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 2004 International Conference on Pattern Recognition (ICPR)*, volume 1, pages 765–768, Aug. 23-26 2004.

[4] T.-S. Lee, E.-M. Lee, H.-T. Park, Y.-K. Kwag, S.-S. Lim, J.-H. Baek, and B.-W. Hwang. Implementation of traffic flow measuring algorithm using real-time dynamic image processing. In *Proceedings of 2003 IEEE International Conference on Computer Vision Systems (ICVS)*, pages 78–87, Graz, Austria, Apr.1-3 2003.

[5] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings of 2002 IEEE International Conference on Image Processing (ICIP)*, volume 1, pages 900 – 903, New York, 2002.

Table 2. Synthesis results for primary modules.

	Logic Elements	Memory bits	Memory Blocks
Feature Engine Datapath and Control	1582	0	0
Feature Engine Data RAMs	45	31284	10
Feature Engine Block RAM	4173	103680	64
Integral Computer Datapath and Control	524	0	0
Integral Computer RAMs	70	34633	21
Total	6394	197226	95

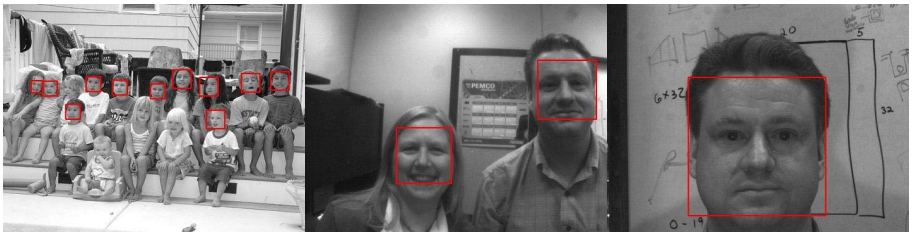

Figure 11. Representative detection results.

[6] H. Nozaki, Y. Motoki, H. Hibino, T. Maeda, and T. Ohta. Digital camera. In *US Patent Application Publication 0,088,538*, 2005.

[7] S. Paschalakis and M. Bober. A low cost FPGA system for high speed face detection and tracking. In *Proceedings of 2003 IEEE International Conference on Field-Programmable Technology (ICFPT)*, pages 214–221, Dec. 14-17 2003.

[8] H. Schneiderman and T. Kanade. Object detection using the statistic of parts. *Int. J. Computer Vision*, 56(3):151–177, Feb. 2004.

[9] M. Sen, I. Corretjer, F. Haim, S. Saha, J. Schlessman, S. S.Bhattacharyya, and W. Wolf. Computer vision on FPGAs: design methodology and its application to gesture recognition. In *Workshop, 2005 IEEE International Conference on Computer Vision (CVPR)*, San Diego, CA, Jun. 21-26 2005.

[10] T.Theocharides, G.Link, N.Vijaykrishnan, M. Irwin, and W.Wolf. Embedded hardware face detection. In *Proceedings of 2004 IEEE International Conference on VLSI Design (ICVLSI)*, pages 133–138, 2004.

[11] P. Viola and M. J. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of 2001 IEEE International Conference on Computer Vision (CVPR)*, volume 1, pages 511–518, Hawaii, Dec.8-14, 2001.

[12] P. Viola and M. J. Jones. Robust real-time object detection. *Int. J. Computer Vision*, 57(2):137–154, 2004.

[13] M.-H. Yang, D. Kriegman, and N. Ahuja. Detecting faces in images: A survey. *IEEE Trans. Pattern Anal. Machine Intell.*, 24(1):34–58, Jan. 2002.

[14] G. Ye, J. Corso, D. Burschka, and G. D. Hager. VICs: A modular vision-based HCI framework. In *Proceedings of 2003 IEEE International Conference on Computer Vision Systems (ICVS)*, pages 257–267, Graz, Austria, Apr.1-3 2003.

[15] W. Zhao, R. Chellappa, and P. Phillips. Face recognition: A literature survey. *ACM computing Surveys*, 35(4):399–458, Dec. 2003.