A DSP-BASED APPROACH FOR THE IMPLEMENTATION OF FACE RECOGNITION ALGORITHMS

A. U. Batur
Center for Signal and Image Proc.
Georgia Inst. Of Technology
Atlanta, GA

B. E. Flinchbaugh
Imaging and Audio Lab.
Texas Instruments
Dallas, TX

M. H. Hayes III
Center for Signal and Image Proc.
Georgia Inst. Of Technology
Atlanta, GA

ABSTRACT

Face recognition is an important part of today's emerging biometrics and video surveillance markets. Recent years have witnessed an exploding interest in the development of face recognition algorithms and products. Currently, face recognition systems are usually implemented on general purpose processors. As face recognition algorithms move from research labs to the real world, power consumption and cost become critical issues. This motivates searching for implementations using a digital signal processor (DSP). Our goal in this paper is to explore the feasibility of implementing DSP-based face recognition systems. To achieve this goal, we implement a fully automatic face recognition system on Texas Instruments' TMS320C6416 DSP, profile performance, and analyze opportunities for optimization. The results of our experiments demonstrate that a generic C implementation with a modest C level optimization effort results in a face recognition software prototype that has low CPU and memory requirements. Hence, it appears that well-optimized face recognition implementations on DSPs can be an effective choice for embedded face recognition products.

1. INTRODUCTION

Biometrics and automatic video surveillance are two emerging markets that are attracting an increasing interest from the research community and the industry. An important technology for these markets is automatic face recognition, which is the task of identifying a person based on an image of his or her face. Although face recognition has been a research area for almost thirty years, there has been a significantly increased research activity since 1990, which has resulted in the development of successful algorithms and the introduction of the first commercial products. Currently, the common hardware choice for the implementation of face recognition systems is a general purpose processor. However, the cost and power issues that come with general purpose processors motivate searching for other platforms. Our goal in this paper is to explore the possibility of implementing face recognition systems on DSP-based platforms. DSPs can provide attractive opportunities for low-cost and low-power implementation of face recognition algorithms, and these advantages may be critical for high volume deployment of face recognition systems in real world settings. To determine the feasibility of implementing DSP-based face recognition systems, we implement a fully automatic face recognition system on Texas Instruments' TMS320C6416 DSP, profile performance, and analyze opportunities for optimization. We determine the CPU and memory requirements for the final prototype system, and we identify important performance bottlenecks. Our results can give an overview of the advantages and challenges of implementing face recognition systems on DSPs.

In the next section, we first describe the face recognition algorithms we implemented. This description can make it easier for the reader to interpret the performance results we have obtained since different algorithms may have different complexities. Then, in Section 3, we give a detailed description of our implementation. Finally, in Section 4, we show the performance profile of our prototype system and discuss the performance bottlenecks we have identified.

2. SYSTEM DESCRIPTION

The goal of face recognition is to determine the identity of an individual based on a still image or video sequence of his or her face. Based on whether the input is a still image or a video sequence, face recognition takes different approaches, each of which has its advantages and challenges. Figure 1 shows the block diagrams of two possible approaches to face recognition. For simplicity, the block diagrams assume that there is a single face in the given image or video sequence. In case multiple faces exist, the systems should process each of them separately.

With a still image input, the system whose block diagram is shown in part (a) of Figure 1 first finds the location of the face with a face detection module. Then, it searches for specific facial features, usually the eyes, to register the face image. Finally, the registered image is normalized, and a classification algorithm determines the identity of the person. Note that searching for the face and the features in still images is a computationally intensive task. In the case of a video sequence input, the system whose block diagram is shown in part (b) of Figure 1 finds and tracks the face using video information. Since motion is a very important clue, a video sequence can significantly simplify face detection and feature localization stages. For example, the movement of the face and the blinks of the eyes can quickly give an idea about where the face and the eyes are. Having this information, the normalized face image can be easily obtained and sent to the classification algorithm. If a face detection and tracking algorithm that utilizes video information is not available, then the system should somehow select some specific frames from the video according to some criteria, and send them to the face recognition block. In this case,

This work has been performed at Texas Instruments, Dallas, TX

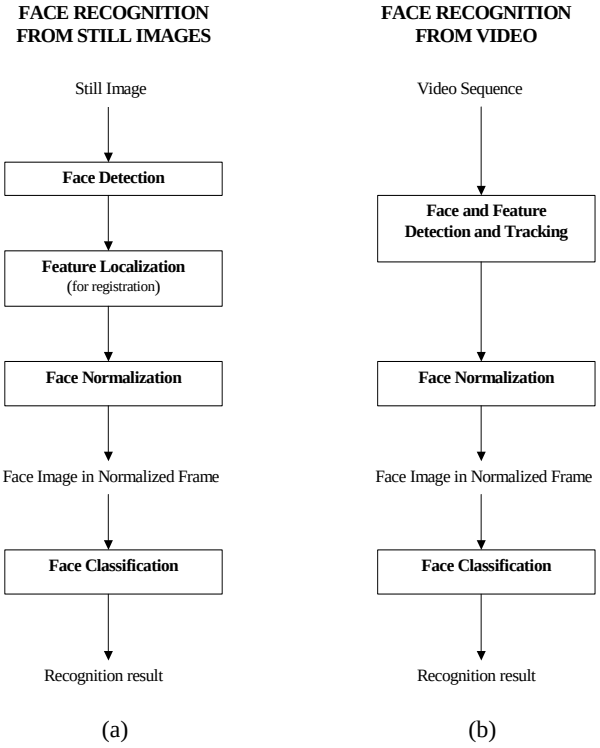

Fig. 1. Face Recognition Block Diagram.

the video problem boils down to the still image problem where no motion information is used.

We implemented a fully automatic DSP-based face recognition system that operates on still images. Our implementation follows the block diagram shown in part (a) of Figure 1. In this section, we shortly review the algorithms we implemented. Our system consists of a face detection block, an eye localization block, a face normalization block, and two face classification blocks. A face recognition system clearly needs only one face classification block, but our goal was to explore DSP feasibility for different approaches. The references for the algorithms we implemented are [1], [4], [5], and [7]. For a detailed discussion of these algorithms and their detection or recognition rates, please refer to these papers.

We can give a short description of our face recognition system as follows: For face detection, we used the probabilistic visual learning approach proposed by Moghaddam and Pentland [5]. According to their approach, face images are modeled as a multi-dimensional Gaussian distribution that is estimated with the help of a Karhuenen Loeve Transform (KLT) based dimensionality reduction. To detect faces in still images, blocks at different scales and locations are extracted from the image, and their probabilities of being a face is calculated using the density mentioned above. Since searching a large image at multiple scales and locations is a computationally intensive task, we tried to decrease the search space with the help of a rule proposed by Kotropoulos and Pitas [4]. Assuming that there is a single face in a given image, Kotropoulos and Pitas suggest that abrupt changes in the horizontal and vertical profiles of the image correspond to head boundaries. The horizontal profile is obtained by averaging the pixels at each column, and the vertical profile is obtained by averaging the pixels at each row. However, in the general case, if the person is in front of an arbitrary background, it is rarely possible to find the head boundaries with this approach because there are other abrupt changes caused mainly by the background. But, even in this case, we found the rule still useful to decrease the search space for the face. In the horizontal profile, we find the first and last abrupt changes and assume that at least half of the face is located between these two boundaries. If there are false detections due to the background, this just increases our search space without causing us to miss the face. Similarly, in the vertical profile, we find the first abrupt change, and assume that the face is located below that upper boundary. We search the space between these boundaries at multiple scales and locations using the method proposed by Moghaddam and Pentland that we have explained above. After the location of the face is found, eye localization is performed, where we search for the two eyes inside the face at multiple scales and locations. We again use the density estimation technique proposed by Moghaddam and Pentland, this time to model the distribution of the eyes. After we find the eyes, we rotate the face image to make the eyes horizontal, crop it to exclude the background, and decimate it down to a size of 128x128. We call these steps face normalization. An illustration of the face detection, eye localization, and face normalization steps is shown in Figure 2.

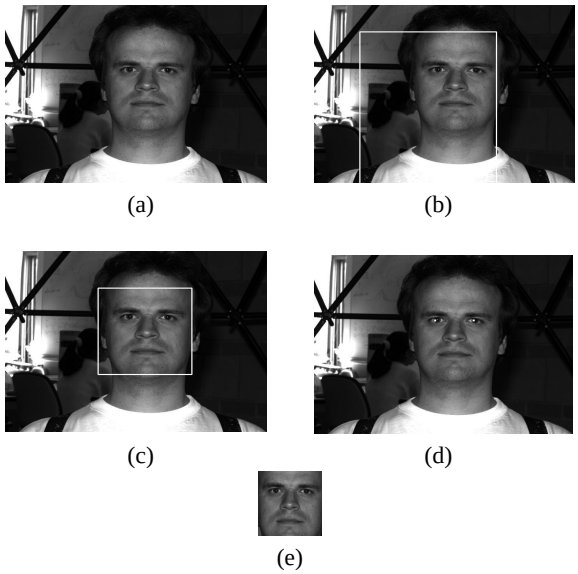

Fig. 2. : Illustration of the face detection, eye localization and face normalization stages. (a) Original image (b) The box shows the reduced search space for the face after the rule-based technique is applied. At least half of the face is assumed to be inside these boundaries. (c) Result of the face detection stage (d) Result of the eye localization stage (e) Normalized face image

After the face image is normalized, we send it to a face classification algorithm that compares it to a database of known people and returns the most likely person. We implemented two different face classification algorithms. The first one is the well-known eigenfaces algorithm proposed by Turk and Pentland, which is considered to be a baseline algorithm for face recognition [7]. According to this approach, face images are first projected into a subspace that is obtained by performing principal component analysis

on the training images. Then, recognition is performed by minimum distance classification. The second classification algorithm we implemented is the segmented linear subspaces algorithm proposed by Batur and Hayes [1]. This algorithm's primary goal is to perform reliable face recognition under varying illumination conditions. According to this approach, each person's face images with a fixed pose under varying illumination are modeled with a segmented linear subspace model, and recognition is performed by computing the distance of the image to the subspace models in the database.

We first implemented the training and recognition procedures for the complete face recognition system in MATLAB, and then, we tested the system using a subset of Yale Face Database B that contains a total of 300 frontal images of 10 people where the lighting direction changes between 0 and 50 degrees [3]. For each person, we used 5 images for training, and the remaining 25 images for testing. For face detection and eye localization, we used 15 dimensional subspaces to estimate the multi-dimensional Gaussian densities. For eigenfaces, we used a 30 dimensional subspace representation, and for the segmented linear subspaces, we used 4 dimensional subspaces with 64 regions. For the fully automatic system, the recognition rate with the eigenfaces classification was 88%, and the recognition rate with the segmented linear subspaces classification was 93%. However, our purpose in this paper is not to evaluate the detection and recognition rates of the specific algorithms we implemented. More detailed information about the performances of these algorithms can be found in their respective papers. After we finalized our algorithms in MATLAB, we started the C implementation of the face recognition system on a DSP. We kept the training procedures in MATLAB because training is usually done offline in controlled environments.

3. IMPLEMENTATION ON A DSP

We implemented the fully automatic face recognition system described in Section 2 on a Test Evaluation Board that contains Texas Instruments' TMDX320C6416 fixed-point DSP and 16 MB of external memory. The DSP runs at 500Mhz and has a two level internal memory architecture. The first level contains a program and a data memory that are 16KB each, and the second level contains a 1024KB memory, called L2. The first level memories can only be used as cache while L2 can be configured as partial static RAM and partial cache. In our implementation, we configured L2 as 256KB cache and 768KB static RAM, which is the configuration with the largest possible amount of cache. We chose this configuration because our system processes a lot of data, which makes the external memory accesses a performance bottleneck, and a large cache increases the efficiency of internal memory usage significantly.

Our original implementation was a double-precision, floating-point generic C code. We compiled it using Texas Instruments' Code Composer Studio C compiler with the optimization options turned on. It consisted of a face detection block, an eye localization block, a face normalization block, and two face classification blocks. Considering that it took around 2 minutes to recognize a single face image, we concluded that this initial generic C implementation was not satisfactory in terms of computation time. Therefore, we performed various C-level optimizations to increase the speed. The significant gains we achieved as a result of these optimizations proved that some sort of optimization effort over generic C code is clearly necessary and is well-worth the effort. In the next section, we first describe the performance bottlenecks we identified throughout our tests, and then, for each of these bottlenecks, we explain the C level optimization tasks we performed to increase the performance. We believe that most of these bottlenecks are not specific to the algorithms we selected, and they can in general apply to the DSP-based implementation of other face recognition algorithms.

3.1. C-level optimizations

The most important performance penalty that our generic C code suffered was due to the overhead of floating-point computations on a fixed-point DSP. Therefore, our initial optimization task was to convert computationally intensive parts of our code to fixed-point arithmetic. The most computationally intensive operations were subspace projections that were computed throughout the face detection, eye localization, and face classification stages. Especially during the search for the face and the eyes at multiple scales and locations, many subspace projections were needed to find the probabilities, and these projections dominated the computational load of the face recognition system. Converting the computations of our detection and classification algorithms to fixed-point was quite straightforward, and the resulting loss in computational accuracy did not seem to be significant since the recognition rates remained exactly the same after the conversion.

Another significant performance penalty for our system was due to not effectively utilizing the parallel computation capabilities of the DSP. Computationally intensive parts of face detection and recognition algorithms are usually large vector-matrix operations that are inherently parallel. Well-known algorithms such as [5], [7], [2], and [6] can be given as examples. Therefore, a hardware platform that specifically facilitates efficient computation of these vector-matrix operations can significantly improve the performance of a face recognition system. The TMS320C6416 provides special instructions for packed data processing to optimize such inherently parallel operations. In fact, the associated DSP library contains assembly-optimized routines that exploit parallelism for some common vector operations. In our system, we used functions from this library to perform subspace projections and vector length calculations. At certain places, we used compiler intrinsics to access special DSP instructions directly from C without switching to assembly to implement fixed-point arithmetic efficiently. Conversion to fixed-point, use of the optimized routines from the DSP library, and use of the intrinsics provided a factor of fourteen increase in the speed.

Another bottleneck for performance was the external memory accesses. Face recognition systems in general process a lot of image data. Storing and accessing this data would probably be the most dominant bottleneck in DSP-based face recognition implementations. In our case, the cache significantly helps to decrease this penalty, but the performance can still be improved by an optimized allocation of data into the internal and external memories. We placed the face detection and eye localization subspaces and other frequently used data into the internal memory, which provided a factor of two increase in speed. The eigenfaces, the segmented linear subspaces, and the program code were placed in the external memory due to their large sizes.

Finally, we compiled our code using the optimization options of the C compiler.

The optimizations we explained above are clearly not complete, and the code can be further optimized to achieve even more gains. We can propose a few major areas for improvement. First

	Number of Cycles ($x10^6$)	Computation Time (CPU at 500 Mhz)	Memory Consumption for Data
Face Detection	1161	2.32 sec.	∼392 KB
Eye Localization	585	1.17 sec.	∼436 KB
Face Normalization	56	0.11 sec.	∼32 KB
Face Classification (Eigenfaces)	18	0.04 sec.	∼1055 KB
Face Classification (Segm. Lin. Sub.)	22	0.05 sec.	∼2064 KB

Fig. 3. Performance profile of the prototype system

of all, to increase the memory access performance and to decrease the memory consumption, DMA can be used, and memory banking and data alignment issues can be addressed. In addition to this, all of the code can be converted to fixed-point to avoid the floating-point overhead completely, and the critical loops in the code can be better organized for software pipelining. Finally, some parts of the code can be optimized at the assembly level for maximum performance.

4. PERFORMANCE PROFILE

We profiled our face recognition system on the DSP by running the recognition software on a 480x640 image that contains a single face. The database we used had 10 people. For the performance profiles, the images processed by the system were available in internal or external memory. Since the rule based approach we used for decreasing the search space for face detection causes variability in computation time, we averaged the performance results over a certain number of input images. The resulting CPU and memory requirements are shown in Figure 3.

A quick look at these results reveals that the face detection and eye localization blocks consume most of the computation time, and the face classification blocks consume most of the memory. These results are expected since searching for faces and features in still images at multiple scales and locations is known to be a computationally intensive task, and the classification blocks have to store the subspace models and the face databases which are large in size. Note that an increase in database size will linearly increase the CPU and memory requirements of the classification blocks. Our implementation follows the still image processing approach shown in part (a) of Figure 1. Therefore, based on the results shown in Figure 3, we conclude that it takes around 3.7 seconds to find and recognize a single face in a 480x640 still image. Most of this time is spent during the face detection and eye localization stages. Hence, choosing faster algorithms for these stages can increase the recognition speed significantly. Also, an implementation that uses video information can speed up the face detection and eye localization stages by using motion information, which can further decrease the total recognition time.

The memory consumption of the face recognition algorithms can be the single most important issue in DSP-based applications. The classification blocks are the critical components for this problem. Memory issues can make the allocation of large databases on the DSP impractical. This can motivate an approach where the initial stages of the face recognition system, until the end of the dimensionality reduction are performed on the DSP, and the final stage where the feature vector is compared to the database is performed at a central server. This approach has the advantage of decreasing the on-chip requirements for memory. Also, maintaining a large database at a central location can be advantageous for some applications.

Finally, a trade-off is clear when we compare the two classification algorithms we implemented. Eigenfaces classification is faster, consumes less memory, and, as we have mentioned in Section 2, provides a lower recognition rate than the segmented linear subspaces method. Similar trade-offs would probably exist for all face classification algorithms.

5. CONCLUSION

The results we have shown in the previous section demonstrate that a generic C implementation and a modest C level optimization effort results in a face recognition system with low CPU and memory requirements on a DSP. The MATLAB, C, and optimized C programs described in this paper were implemented in less than three person-months in a summer research project at Texas Instruments. Keeping in mind that further optimizations can produce even lower CPU and memory requirements, DSP-based implementations appear to be a cost- and power-effective choice for embedded face recognition products.

6. ACKNOWLEDGEMENTS

The authors would like to thank Ram Sathappan and Oliver Sohm for their valuable comments.

7. REFERENCES

[1] A. U. Batur and M. H. Hayes. ”Linear Subspaces for Illumination-Robust Face Recognition,” *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp.296-301, 2001.

[2] P. Belhumeur, J. Hespanha, and D. Kriegman. ”Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 19, No. 7, pp. 711-20, 1997.

[3] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. ”From Few to Many: Illumination Cone Models for Face Recognition Under Variable Lighting and Pose,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 23, No. 6, pp. 643-60, 2001.

[4] C. Kotropoulos and I. Pitas. ”Rule-Based Face Detection in Frontal Views,” *Proc. Int’l Conf. Acoustics, Speech and Signal Processing*, Vol. 4, pp. 2537-2540, 1997.

[5] B. Moghaddam and A. Pentland. ”Probabilistic Visual Learning for Object Representation,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.19, pp. 696-710, July 1997.

[6] K. K. Sung and T. Poggio, ”Example-Based Learning for View-Based Human Face Detection,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 20, No. 1, pp. 39-51, Jan. 1998.

[7] M. A. Turk and A. P. Pentland. ”Face Recognition Using Eigenfaces,” *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 586-91, 1991.