International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Performance Analysis and Improvement of Face Recognition Algorithms on TMS320C64x

S. Chitra

Assistant Professor, ECE, Bharath University, Chennai

ABSTRACT:
Face recognition is an important part of today's emerging *biometrics* and *video surveillance* markets. As face recognition algorithms move from research labs to real world products, power consumption and cost become critical issues, and DSP-based implementations become attractive. Our goal in this research project was to evaluate the CPU and memory requirements of face recognition algorithms on the TMS320C64x platform to determine the feasibility of implementing DSP-based face recognition systems. The results of our project 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 onTMS320C64x, and runs with high speed to enable real-time applications. Therefore, well - optimized face recognition implementations on TMS320C64x are an effective design choice for embedded face recognition systems.

Keywords: Face Recognition-Biometrics-DSP-Optimization

1. Introduction

Biometrics and automatic video surveillance are two emerging markets that are attracting increasing interest from the research community and the industry. An important application in these markets is automatic face recognition, which is the task of identifying a person based on an image of his or her face. Face recognition has been a research area for almost 30 years, with significantly increased research activity since 1990. This has resulted in the development of successful algorithms and the introduction of commercial products. As face recognition algorithms move out of the research labs and into real world applications, power consumption and cost become critical issues. This motivates searching for DSP-based implementations. For a low cost, high volume deployment of face recognition systems, DSPs offer a natural platform.

The goal of our research project was to evaluate the CPU and memory requirements of face recognition algorithms on the TMS320C64x platform. Understanding these requirements will assist in determining the feasibility of implementing DSP-based face recognition systems. To achieve our goal, we implemented a fully automatic face recognition system on TMS320C6416, profiled the performance and analyzed opportunities for optimization.

In the next section, we give an over review of the face recognition problem. Then, in section 3, we describe two DSP-based face recognition applications that we considered in this project. In section 4, we review the specific algorithms that we implemented. Finally, in sections 5 and 6, we describe our DSP-based implementation and show the performance profile.

2. Face Recognition Overview:

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. There are two different modes of operation for a face recognition system: authentication and identification. In the authentication mode, the system accepts or rejects the claimed identity of the individual. In the identification mode, the system compares the face image to a database of known people, and returns the most likely identity or identities. 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 system should work on 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 that case, the video problem boils down to the still image problem where no motion information is used.

3. Examples of DSP-Based Face Recognition Systems

In this project, we considered two exemplary application scenarios for a DSP-based face recognition system.

3.1 Face Recognition Camera

In this scenario, a fully automatic face recognition system is implemented on a single DSP that is integrated into a camera or other image/video end equipment. Such a system can process still images or video sequences. This approach would be convenient for small database applications since storing and maintaining a very large database on each camera might not be practical. A face recognition camera could be applied, for example, in retail stores to look for a number of known shoplifters who might be in the area.

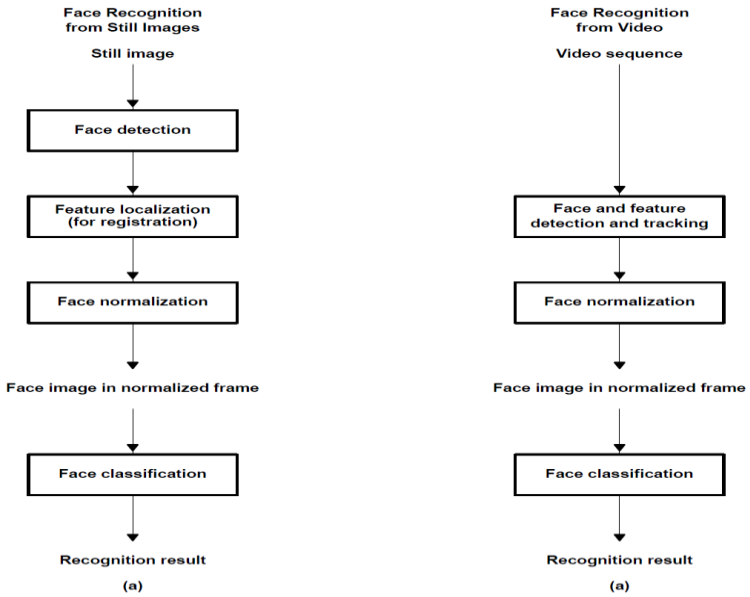

Figure 1. Two Approaches to Face Recognition:
(a) face recognition from still images, (b) face recognition from video

3. PreProcessor2 Face Recognition

In this scenario, a DSP that is integrated into a camera processes still images or video, and obtains the normalized face image. Then, this normalized image is transferred to a central server for face classification. The server compares the image to a large database of faces that is conveniently stored and maintained at a centralized location. An application for a face recognition pre-processor could be a badge reader unit that provides a face authentication function to confirm that an employee's face matches the person who is seeking entry in an access control system, using a network to access a secure corporate database of employee faces.

4 System Description

To test feasibility of the scenarios we described above, we implemented a fully automatic DSP-based face recognition system that works on still images. Our implementation follows the block diagram shown in part (a) of Figure 1. In this section, we review the algorithms Our system consists of a *face detection* block, an *eye localization* needs block, a *face normalization* block, and two *face classification* blocks. A face recognition system clearly only one face classification block, but our goal was to be able to compare different algorithms. The references for the algorithms we implemented are [1], [2], [3], and [4]; 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.[1]. 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 are 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 back ground, 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 and Figure 3. 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 [2], which is considered to be a base line algorithm for face recognition. According to this approach

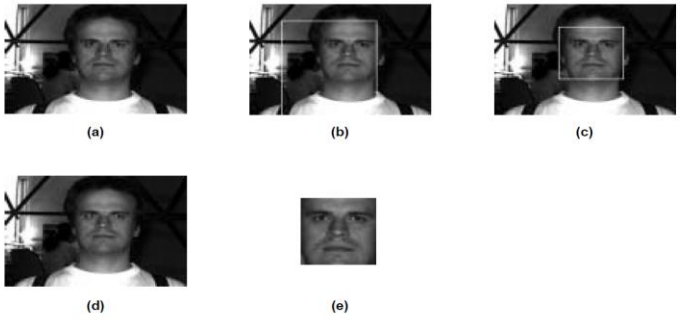

Figure 2. Face Detection, Eye Localization and Face Normalization Stages: (a) original image, (b) The box shows the reduced search space for the face after the rule-based technique is applied. At least half of the face is assumed to be inside these boundaries., (c) result of the face detection stage, (d) result of the eye localization stage, (e) normalized face image

Figure 3. 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

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 [3].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 enrollment and recognition procedures for the complete face recognition system in Matlab, and then, we tested the system on 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 [7]. This database can be obtained from the web page located at [8]. 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%. Our goal in this project is not to evaluate the detection and recognition rates of the specific algorithms we implemented, and 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 TMS320C64x DSP. We kept the enrollment procedures in Matlab because enrolling is usually done offline in controlled environments.

5 Implementation on C64x

We implemented the fully automatic face recognition system described in section 4 on a C6416 TEB that contains a TMDX320C6416 fixed-point DSP running at 500 Mhz and an external memory of size 16MB. The DSP has a two level internal memory architecture where 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. For more information about this issue, please refer to the application note titled *Cache Usage in High-Performance DSP Applications With the TMS320C64x* (SPRA756) Our original implementation on the C64x DSP was a floating-point generic C code compiled with the Code Composer Studio C compiler. 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 generic C implementation on C64x 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.

5.1 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. To make efficient use of the 16-bit multiplier C64x has, we tried to use Q.15 as much as possible. Q.15 is a 16-bit fixed-point representation where the last 15 bits are used as the fractional part. 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 C64x's parallel computation capabilities. Computationally intensive parts of face detection and recognition algorithms are usually large vector-matrix operations that are inherently parallel. Well-known algorithms such as [1], [2], [5], and [6] can be given as examples. These large vector operations can be very well optimized on C64x that provides a powerful architecture and special instructions for packed data processing. In fact, C64x DSP library contains assembly-optimized routines for some common vector operations. In our system, we used functions from this library to perform subspace projections and vector length calculations. For more information about this library, please refer to TMS320C64x DSP Library Programmer's Reference. At certain places, we used *C64x i*ntrinsics to access special C64x instructions directly from C, without switching to assembly. The intrinsics we used were _sshvl and _mpylir, which helped us to implement fixed-point arithmetic efficiently.Conversion to fixed-point, use of the optimized routines from the C64x DSP library, and use of C64x intrinsics provided a factor of 14 increase in the speed. Another bottleneck for performance was the external memory accesses. Although the cache helps to decrease this penalty, the performance can still be improved by an optimized allocation of data into the internal and external memories. We placed the face detection and eyelocalization subspaces and other frequently used data into the internal memory, which provided us *a factor of 2* increase in speed. The eigen faces, the segmented linear subspaces, and the program code were placed in the external memory due their large sizes. Finally, we compiled our code using –o3 –pm optimization options of the Code Composer Studio C compiler, and made sure that we did not use –ss option which decreases the execution speed. The optimizations we explained above are clearly not complete, and the code can be further optimized to achieve even more gains. We can propose four major areas for improvement. First, DMA can be used to increase the efficiency of internal memory usage. Second, all of the code can be converted to fixed-point to avoid the floating point overhead completely. Third, 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.

6 Performance Profile

We profiled our face recognition system on C64x by running the recognition on a 480x640 image that contains a single face. The database we used had 10 people. We assumed that the data that would be processed by the system was 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 Table 1. 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. Now, let us discuss these results from the point of view of the face recognition approaches we have explained in section 2. Our implementation follows the still image processing approach shown in part (a) of Figure 1. Therefore, based on the results shown in Table 1, we can conclude that it takes less than 4 seconds to recognize a single face from a 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. Although we did not implement the video based approach shown in part (b) of Figure 1, we can make some comments about it based on the face normalization and classification blocks we have implemented. We believe that it is possible to implement a real time face and facial feature detection and tracking system on C64x. If such as system is available to process the video, the location of the face and facial features can be known at any time. Then, according to the results shown in Table 1, normalization and classification takes less than 1 second to complete. This can make it possible to implement a very fast face recognition system on C64x. The above comments are all related to the face recognition camera application we have described in section 3.1. The difference of the face recognition pre-processor application shown in section 3.2 is to move the classification block to the central server. Large memory requirements of the classification blocks shown in Table 1 may motivate such an approach. Finally, if we look at the performance profiles of the two classification algorithms, we notice a trade off between the recognition rate and the computational complexity. *Eigen faces* classification is faster, consumes less memory, and, as we have mentioned in section 4,provides a lower recognition rate than the segmented linear subspaces method. Similar tradeoffs would probably exist for all face classification algorithms.

Table 1. Performance Profile

	Number of Cycles (x 10^6)	Computation Time (With a CPU at 500Mhz)	Memory Consumption for Data
Face detection	1161	2.32 seconds	~ 392 KB
Eye localization	585	1.17 seconds	~ 436 KB
Face normalization	56	0.11 seconds	~ 32 KB
Face classification (eigenfaces)	18	0.04 seconds	~ 1055 KB
Face classification (segmented linear sub.)	22	0.05 seconds	~ 2064 KB

7. Conclusion

The results, we have shown in the previous section demonstrate that a generic C implementation with a modest C level optimization effort results in a face recognition system with low CPU and memory requirements on C 64x.There could be further optimizations to achieve even lower requirements, it appears that DSP - based implementations can be a cost and power-effective choice for embedded face recognition products.

8. Acknowledgements

I would like to thank Dr.Caroline Britto who encouraged me to publish a paper and also i express my thanks to my family members .

9. References

1. 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.
2. M. A. Turk and A. P. Pentland. “Face Recognition Using Eigenfaces,” *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 586-591, 1991.
3. 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.
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. 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. 711-720, 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. 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. 643-660, 2001.