Development and Implementation of a New Real-Time Face Recognition Algorithm on an Embedded Hardware

Gökhan Saroğlu
Mustafa Tok
Ufuk Bozkurt

Assoc. Prof. Dr. Türker İnce

May 30, 2013

Abstract

We aim to develop a real-time system which is responsible for identifying people from their faces and executing predefined operations. The methods used in the proposed system are motivated by the areas of computer vision and embedded programming. We have examined different algorithms, not only for face recognition but also for face detection, in terms of their performance, time and space complexities to find the best solution for real-time embeded implementation.

Among the proposed approaches for face recognition systems, Viola-Jones algorithm for face detection and the eigenface technique for face recognition stand out as the most popular and most successful methods in this field. In this project, our objective is first understanding how these above mentioned techniques work and analyze their performance. Next, we shall adapt these powerful techniques for our proposed two educational real-time system solutions.

Contents

List of Figures

List of Tables

1 Introduction

Object recognition and detection is a very important subject for the area of computer vision since 1980s. Face recognition is a vital part of the object recognition area and it becomes significantly important for the state-of-art technologies. The problem of automatic recognition of human faces can be defined as detecting and identifying faces in given still images or video images by using a stored database of known faces. Face recognition is being used in many different fields such as security, biometrics, entertainment and scientific research.

There are a lot of researches for face recognition techniques. Even though, current machine learning technology have been developed significantly, current systems are still far behind the capabilities of human perception. Hence, recognition of face image in real applications remains an unsolved problem.

The objective of our project is to develop a real-time approach to face recognition and face detection problem for education purposes. Taking the students' attendance automatically in a classroom and accessing and securing the laboratories without using external keys are the proposed applications. In order to develop these systems, face detection and recognition algorithms are needed. Moreover, these algorithms should be run on a computer independent hardware. We have chosen Viola-Jones algorithm [3] for face detection problem and principle component analysis(PCA) based eigenface method [2] for recognition problem. In addition, an embedded hardware is used for real-time implementation. Advantages of the embedded hardware are low-energy consumption, portability, high processing capability and it is cheaper and smaller than a PC.

We have used various kinds of tools in implementation phase. MATLAB and/or C/C++ programming language, Simulink, OpenCV framework and many different benchmark face databases are the examples of some tools of which we have taken advantage. Yale and MIT face image databases [5, 6] are the examples we have used. Beagleboard XM is used for real-time implementation. It contains 1 GHz processing power and 512mb low power DDR RAM. Moreover, it has different interfaces (4 USB ports, 10/100 Ethernet port, Serial port, Stereo audio in/out, S-video (TV out), DVI-D (HDMI)) to connect peripheral devices and internet.

This report explains the basics of our proposed method and description of the algorithms. Moreover, it contains a general introduction for the real-time emmbeded systems. In section 2, introduction to the field and related works

which are face detection, face recognition and real-time embeded systems are presented. Proposed system is explained in section 3. In the next section, experimental results of the mentioned algorithms are studied.

2 Background

In this section, basics of the proposed algorithms are handled. The first subsection is elaborates the our proposed face detection algorithm which is Viola-Jones object detector. Face recognition algorithms are explained in section 2.2. In the last subsection, the background of the real-time embedded implementation is stated.

2.1 Face Detection

There are many face detection methods in the literature whereas the most successful one is the Viola-Jones algorithm. Viola-Jones algorithm has a very expensive training phase. In contrast, algorithm detects the faces very accurately and fast. This algorithm is used in many commercial applications such as digital cameras, web sites, etc.

In training phase, algorithm chooses rectangle haar like features which are suitable for face detection. These features are can be any height and width. In addition, these rectangular areas can be at any place on the image. Therefore, there are approximately 160.000 features in a 24x24 image.

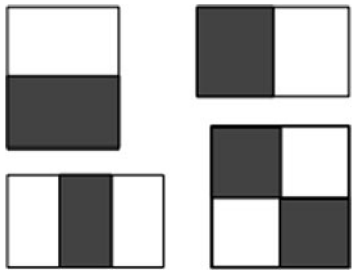


Figure 1: Example for Rectangle Features

The calculation of this features is a simple process. In order to calculate value of a rectangular feature, sum of the pixel values under the black regions is substracted from sum of the pixel values under the white regions. This calculation can be done in constant time with the integral image representation.

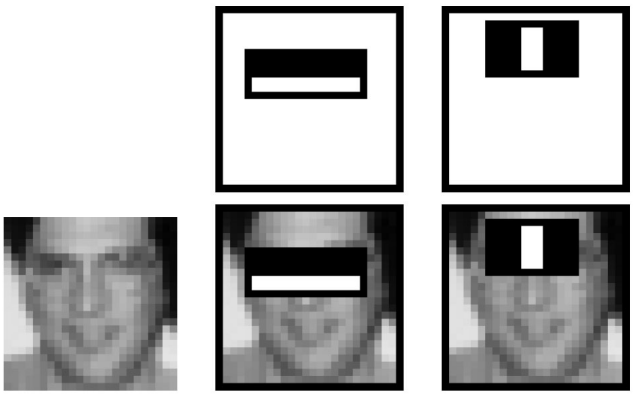


Figure 2: Example for Calculation of Rectangular Features on a Face [3]

Integral image can be defined as follows: Let $f(x, y)$ be the pixel value at (x, y) in original image and $g(x, y)$ be a pixel value of integral image. Then

$$g(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} f(x, y)$$

Thus, sum of the pixels of a rectangular area can be calculated with four reference points in constant time.

Even if a rectangular feature can be calculated in constant time it is not suitable to calculate approximately 160.000 features for every 24x24 subwindow on a real-time operation. Therefore, best features must be picked so that we have less number of efficient features. This picking process is done with the AdaBoost machine learning algorithm.

Adaboost algorithm picks the best features and uses them to train strong classifiers. AdaBoost algorithm assigns a threshold for each feature and features becomes to weak classifiers with this thresholds. Thresholds can be calculated in many different ways such as taking mean of the values that rectangular feature produces on face images. Simplified formula for a weak classifier (f_i denotes rectangular feature and θ denotes the threshold):

$$h_i(x) = \begin{cases} 1 & f_i(x) > \theta \\ 0 & otherwise \end{cases}$$

After finding the weak classifiers, it is easy to selecting best weak classifiers according to their true positive and true negative rates. AdaBoost algorithm

constructs strong classifiers with using weak classifiers. A strong classifier is simply a weighted linear combination of the weak classifiers.

$$H(x) = \begin{cases} 1 & \sum_{i=0}^{T} \alpha_i h_i \geq \frac{1}{2} \sum_{i=0}^{T} \alpha_i \\ 0 & otherwise \end{cases}$$

In above formula $H(x)$ is a strong classifier, h_i's are weak classifiers and α_i's are weights. Since $H(x)$ is a classifier it has a threshhold which can be determined in many different ways. Strong classifiers are easy to calculate again but it is again in efficient for a real time operation to calculate all strong classifiers for all sub-windows. Cascade architecture solves this problem in a very clever way.

Strong classifiers are divided in to parts according to their false negative and false positive rates in cascade architecture. After that any sub-window which is rejected by a part of cascade architecture rejected immediately so system avoids from the calculation of the rest of the parts and this decreases the calculation load on the system.

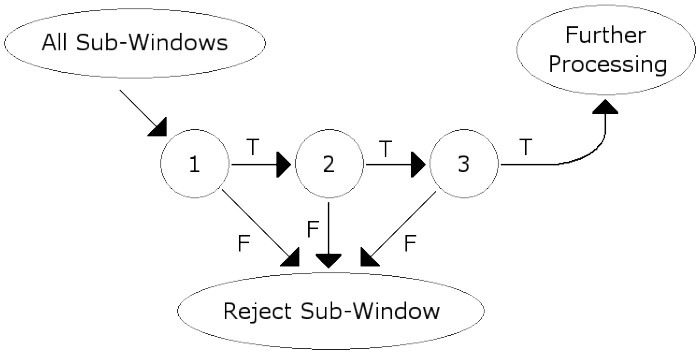


Figure 3: Cascade Architecture Example

2.2 Face Recognition

We have chosen PCA based eigenface approach for face recognition operation. Before eigenface approach, correlation method has been used. However, this method was a in efficient one.

2.2.1 Correlation Method

In this method, classification process is done by measuring the distances between the test image and all training set images. After finding distances, the test image is assigned to the label of the training set image which is the closest to the test image. These distances are measured in the image space and the all images in this space are normalized to have zero mean and unit variance. Therefore, finding the closest training set image to the test image becomes finding training set image that best correlates the test image. Moreover, result of the recognition process becomes independent of the illumination because of normalization.

Since the test image should be correlated with all images in the training set, correlation is a computationally expensive method. In addition, this process requires a huge storage area because there should be many different images of each subject to catch a good classification rate.

2.2.2 Eigenface Method

Correlation method is not suitable for real-time face recognition system because of its time and space complexity. One method is to reduce space complexity is using principal component analysis(PCA). Eigenface method is a face recognition technique based on PCA. Eigenfaces are set of vectors which are used in human face recognition problem. This approach was first developed by Sirovich and Kirby [1]. Sirovich and Kirby used eigenface method to represent faces in low dimentions. After this approach, Pentland and Turk [2] developed a face recognition system with eigenface technique.

The eigenface approach consist of two phases, training phase and recognition phase. In training phase, we have a training set of n face images which has size of MxN. First, we transform our training set faces to vectors of size of MNx1. Let this vectors be $\Gamma_1, \Gamma_2, ..., \Gamma_n$. After the transformation, we find the mean of our face vectors(Figure 4). The mean of our face vectors are defined as:

$$\Psi = \frac{1}{n} \sum_{i=1}^{n} \Gamma_i$$

Figure 4: An example of mean face

Then, for each of faces we find their zero means by subtracting the mean vector from their vectors. Thus, zero mean values for each face: $\Phi_i = \Gamma_i - \Psi$ where $i = 1, .., n$. We will use these zero means to construct covariance matrix. First, we construct matrix A such that $A = [\Phi_1\ \Phi_2\ ...\ \Phi_n]$. Afterwards covariance matrix C calculated as $C = AA^T$. Eigenvectors of the covariance matrix gives us eigenfaces of our training set. However, its obvious that matrix AA^T is a very large matrix (MNxMN) and finding its eigenvectors is an inconvenient operation. Instead of finding the eigenvectors of C, we can find the eigenvectors of the matrix $L = A^T A$ which is much more smaller than C. Then, we can use the eigenvectors of L to find the eigenvectors of the C [2].

$$
\begin{aligned}
Lv_i &= \lambda_i v_i \\
A^T A v_i &= \lambda_i v_i \\
AA^T A v_i &= A\lambda_i vi \\
\underbrace{(AA^T)}_{C} A v_i &= \lambda_i A vi \\
C \underbrace{(Av_i)}_{u_i} &= \lambda_i \underbrace{(Av_i)}_{u_i} \\
Cu_i &= \lambda_i u_i
\end{aligned}
$$

Above formulation shows that if v_i is an eigenvector of L then (Av_i) is an eigenvector of C. Therefore, we can define the eigenvectors of the C such that $u_i = Av_i$. Eigenvectors of C are called *eigenfaces*(Figure 5) because they look as faces and consist of eigenvectors. They create a face space U such that $U = [u_1\ u_2 ... u_n]$.

After calculating the face space, we need the pattern vectors to represent each instance. The pattern vectors of each face can be calculated by

Figure 5: Examples of eigenfaces

projecting zero mean faces to face space by using the formula:

$$\Omega_i = U^T(\Gamma_i - \Psi)$$

In order to recognize face image, the pattern vector of new image(Ω_{new}) has to be calculated as $\Omega_{new} = U^T(\Gamma_{new} - \Psi)$ where Γ_{new} is the new image. The important step of recognition is to ensure that the given image is a face image. In order to do this, the image should be reconstructed from face space by using the formula:

$$\Gamma_{reconst} = U\Omega_{new} + \underbrace{(\Gamma_{new} - \Psi)}_{\Phi_{new}}$$

Then, we have to calculate the euclidean distance from the image to its reconstructed form $\epsilon_\epsilon = \| \Gamma_{new} - \Gamma_i \|$ and it is used to determine the image is face or not. After that, the distance from the pattern vector of input image to each pattern vector in training set is calculated by this formula, $\epsilon_i = \| \Omega_{new} - \Omega_i \|$ where $i = 1...n$. It is used to determine the class of a person. The final process of recognition is to compare distances with threshold(θ) which is calculated in training phase.

- The given image is a non-face image ($\epsilon_\epsilon > \theta$).
- The image is a face image and exists in training set ($min(\epsilon_i) < \theta$).
- The image is a face but it is not in the training set ($min(\epsilon_i) > \theta$).

Note that euclidean distance is used as distance metric; however, different metrics can be used.

The pseudo code of algorithm :

input : Training Set [Γ_1 Γ_2 ... Γ_n]
$\Psi \leftarrow$ calculate mean face
$A \leftarrow$ subtract mean face from each instance in training set
$L \leftarrow A^T A$
$eigVectors \leftarrow$ eigenAnalysis(L)
$U \leftarrow$ calculate eigenfaces ($A * eigVectors$)
$\Omega_i \leftarrow$ calculate pattern vectors ($U^T(\Gamma_i - \Psi)$)
$\theta \leftarrow$ calculate threshold
return U, Ω, θ

Algorithm 1: Training algorithm

input : MxN Image
output: Recognized face or false
$\Gamma_{new} \leftarrow$ convert input image to column vector
$\Omega_{new} \leftarrow$ calculate pattern vector ($U^T(\Gamma_{new} - \Psi)$)
$\Gamma_{reconst} \leftarrow$ calculate reconstructed image ($U\Omega_{new} + \Phi_{new}$)
$\epsilon_\epsilon \leftarrow$ calculate distance to reconstructed img($\| \Gamma_{new} - \Gamma_i \|$)
$\epsilon_i \leftarrow$ calculate distances to pattern vectors ($\| \Omega_{new} - \Omega_i \|$)
if $\epsilon_\epsilon > \theta$ **then**
  It is not a face image
else if $min(\epsilon_i) < \theta$ **then**
  face $\leftarrow$ find_index($min(\epsilon_i)$)
  return face
else if $min(\epsilon_i) > \theta$ **then**
  The face is new and not in the training set.
end

Algorithm 2: Recognition algorithm

2.3 Real-Time Embedded Implementation

An embedded system consists of hardware and software to do a dedicated job[7]. Embedded systems are so popular and have an important role in our daily lives. For example, last few years smart televisions are invented and they perform lots of operations independent from a Personal Computer(PC). Many people cannot realize those systems are embedded systems.

Some embedded systems are extremely fast compared to PC because computers are general purpose devices and executing many applications simultaneously. This section contains information about three popular embedded system types: Microprocessor, Microcontrollers and Digital Signal Processors.

2.3.1 Microprocessors

Microprocessor is a chip that is designed to perform arithmetic and logical operations according to instructions. Most of the microprocessors include adding, subtracting and compare instructions inside its instruction set. A typical microprocessor contains; Central Processing Unit(CPU), Arithmetic Logic Unit(ALU), Memory, Instruction Registers, Data Registers and busses. CPU is a structure that is executing all instructions which are given by the program. ALU is a structure which is executing mathematical instructions.

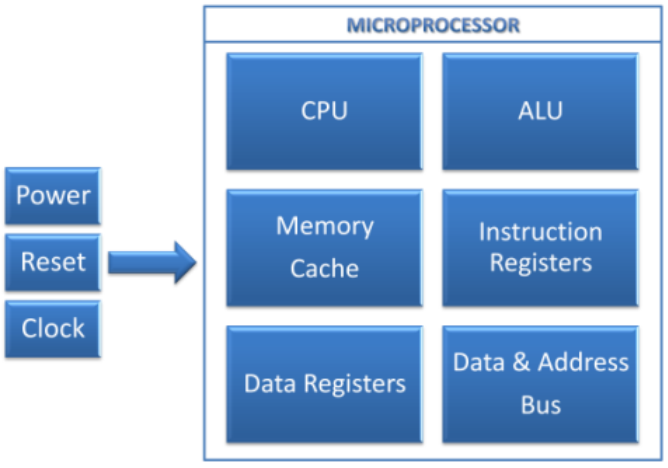


Figure 6: Typical Microprocessor Architecture

Memory is a small area which program and data loads into but main program and data is stored in outside of the microprocessor. Instruction registers are used to store the instruction which is set by the program. Bit processing capability depends on the size of the instruction registers. In each clock cycle, CPU needs to store data inside of the microprocessor. In that case, data registers are used to store data. Busses are used to transfer instructions and data between previously explained structures.

2.3.2 Microcontrollers

Microprocessors can only perform calculations and store results of them but it cannot communicate with other components by itself. Because of that reason microcontrollers are invented.

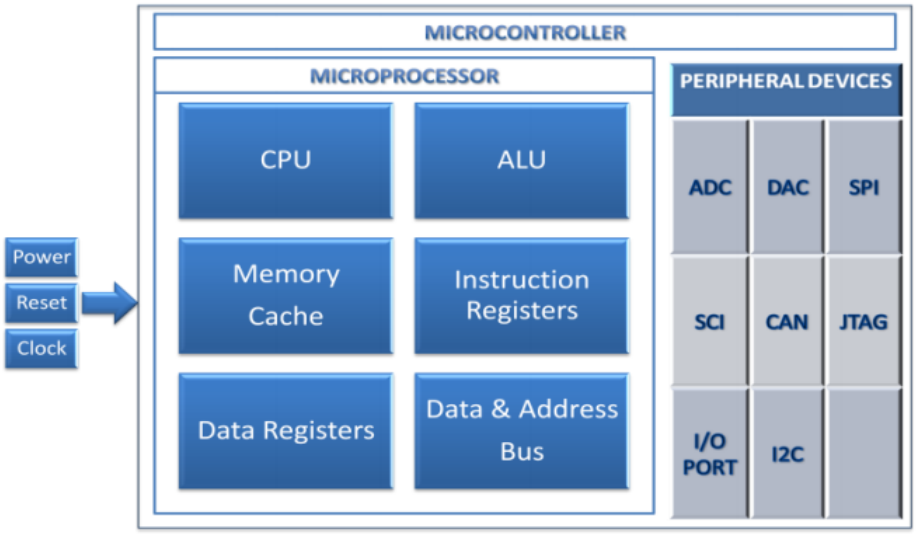


Figure 7: Typical Microcontroller Architecture

Microcontroller is a chip which contains both microprocessor and peripherals. According to their usage area, microcontrollers include different type or peripherals. Analog to Digital Converter (ADC) and Digital to Analog Converter (DAC) are needed in a microcontroller which is designed for audio or video processing. JTAG (A fast real-time communication protocol), Serial Peripheral Interface (SPI) or RS232 Serial Communication Interface (SCI) is needed to communicate with another microcontroller or computer.

2.3.3 Digital Signal Processors

Digital Signal Processor (DSP) is a special Microcontroller which is designed for processing signals.

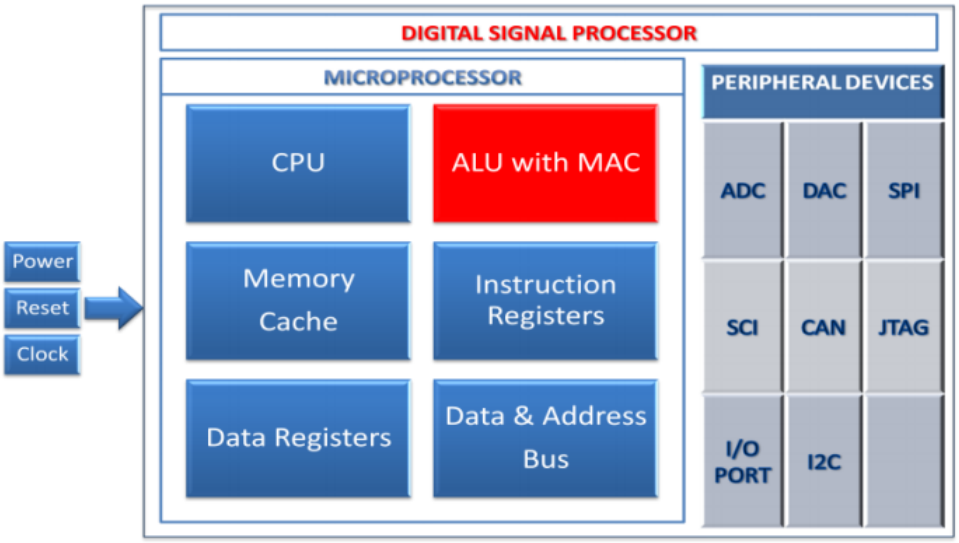


Figure 8: Typical DSP Architecture

In signal processing, filtering, frequency-time domain conversions and convolution are commonly used applications. These operations need multiply and accumulate (MAC) operations. A standard microcontroller performs multiplication as series of addition operation. That means a standard microcontroller performs MAC operations in excessive number of addition operation so MAC operations become very slow in standard microcontroller. However, DSPs contains special MAC unit which is capable of executing same operation in single clock cycle. Because of that reason, Digital Signal Processors are really fast when compared to Microcontrollers.[4]

We have used the Texas Instruments TMS320C6713 DSK board for the system. Specifications of this board can be found in Appendix A.

3 Proposed System

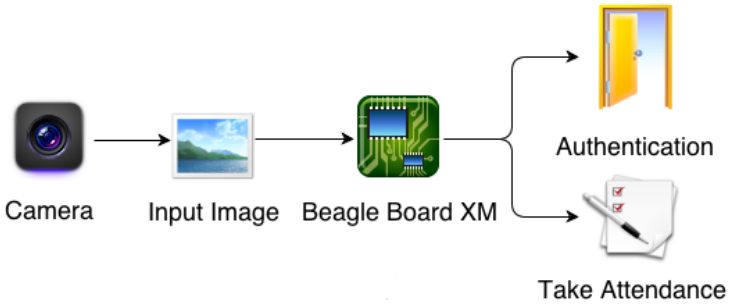


Figure 9: Problem Definition

Main objective of our project is automatically taking attendance and giving permission for entrance to laboratories. In order to achieve these goals, we have designed a system shown in Figure 9 . As shown in Figure 9 , our proposed system has a client-server type architecture. In this system, every client is an embedded hardware located in different classrooms and they are connected to a camera to take the image of students. Purpose of the server is to hold the necessary information that are required for the recognition process and to record the result of the attendance taking process.

First of all, each client needs to get required information from server. Therefore, a socket connection is established between server and client. Then client sends its location to server and requiests the necessary information for recognition process. Later, client reads an image from the camera and starts the detection and recognation process with the information that are sent by server. After that, client returns the result, which will be the recorded to database by the server.

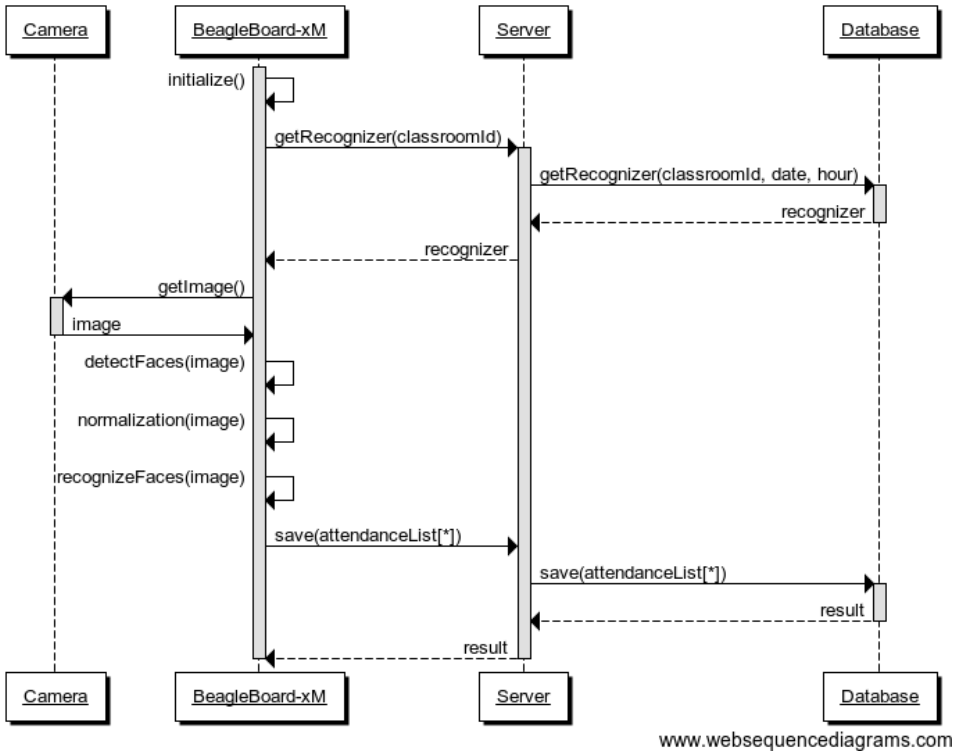


Figure 10: Sequence Diagram

4 Implementation

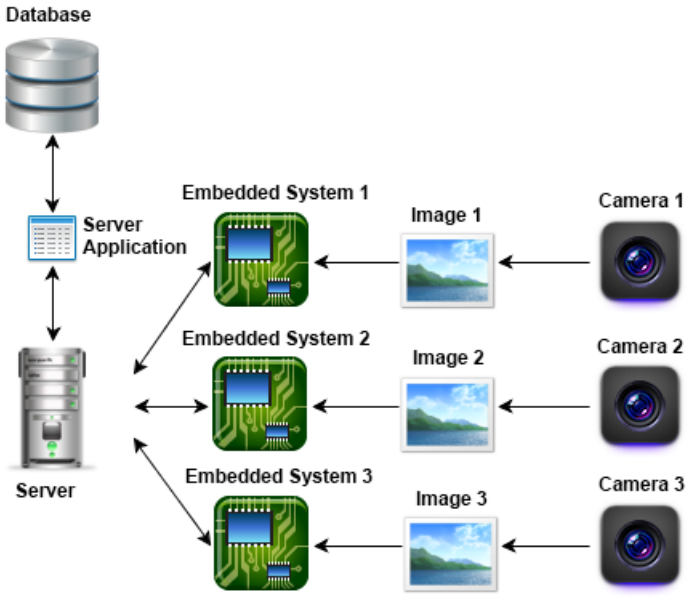


Figure 11: System Design

In order to implement our proposed system, we bought a beagleboard xm as our embedded hardware and we have used C++ and Java as our programming languages. Moreover, we have used Open CV framework to efficiently implement our algorithms. In this section, implementation of the server and the client sides will be explained in detail.

4.1 Server Implementation

Server application is written with Java. The reason for why we chose java is to create flexible and portable server application for our problem. Server determines the logic of the application thats why we want to make it as flexible as possible. Server application uses some well-known java libraries such as Netty[8], Distruptor[9] and BoneCP[10].

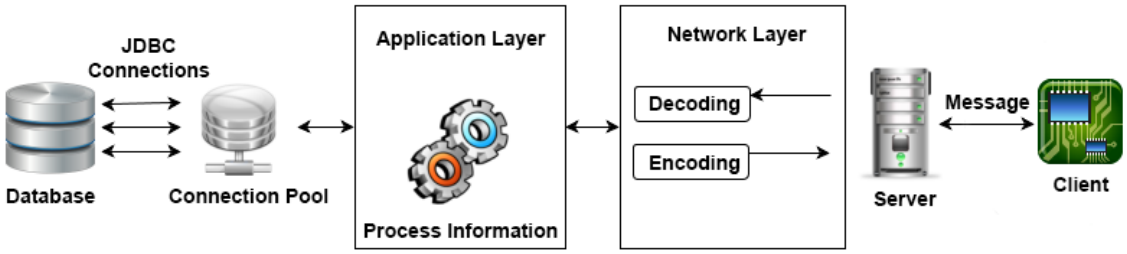


Figure 12: Detailed Server Diagram

Netty library is used for network layer of the server application. It helps to separate network and application layer from each other. That means application logic does not need to know network layer details. Application layer only focuses on the problem. In addition, Netty provides a multi-threaded environment for network layer to communicate with clients.

Distruptor is powerful inter-thread messaging library. It uses a different approach to inter-thread messaging. The classic messaging libraries use blocking queue logic when messages are distributed into threads, but distruptor uses circular queue messaging system without using any lock mechanism. It is really fast library when it is compared with the similar libraries.

Bone CP is JDBC(Java Database Connection) connection pool library. It connects databases, which have JDBC drivers (Oracle, MySQL, PostgreSQL, MSSQL, etc). Connection pools are important for application performance when the multi-threaded environment is concerned. Threads should not wait for idle database connection otherwise performance issues will occur. That is why we are using a connection pool for the server application. Whenever a thread needs a connection, it gets existing connection from the pool after it finishes its job, it gives back to the pool.

Network layer uses thread pool to send and receive messages. Thread pool size is fixed on the application, and it is not depending on how many clients are going to connect to server because using many threads decreases the general server performance. Whenever a packet comes to network layer, network thread extracts the message from packet according to the protocol and it passes it to another thread which executes the logic of the application. Whenever application layer needs to send a message to the client, it sends the message to network layer without considering the network protocol. Net-

work layer encodes the message and sends it to the client. In order to keep everything simple, we implemented string-based protocol but implementing different protocol may increase the efficiency the system.

Application layer is the part of the server that implements the logic of the application. Application layer is responsible for transferring data from database and client. It determines packet types and the action whenever a message came to server. Different implementation of this layer can solve different types of problems. In addition, this layer is responsible for logging the actions that are done in client side. Whenever a client sends a log information, server will save the log into the database for error checking or action logging.

4.2 Client Implementation

Figure 13: Detailed Client Diagram

Our client side contains two hardware components, which are a Beagleboard-xM (BBx) and a USB-Camera. BBx is an embedded hardware powered by an embedded linux operating system. This provides us a high-level software development environment. In order to run our algorithms on BBx, we have benefited some powerful libraries which are OpenCV[11] and Boost[12].

Open Computer Vision(OpenCV) library is an open source Computer Vision library written in C/C++. The reasons why we have used OpenCV are to get frame from camera and detect faces from that frame. Moreover, while developing our recognition algorithm we have used some necessary functions in OpenCV.

Boost is an advanced open source library for C++. Boost complements the missing parts of C++. We have used serialization functionality of boost library to send data on network easily.

4.2.1 Environment Setup

There is a pre-installed Angstrom Linux on SD Card which comes with BBx. However, installed version of distribution is net configured with your requirements. In order to construct a customized and better development platform, we have followed a couple of steps.

1. In order to communicate with BBx we obtained serial cable.
2. We customized and downloaded the new version of Anstrom Operating System from the website[13].
3. We installed the OS image to an SD Card of BBx. The installation is performed by "dd" command in Unix based operating systems and detailed instructions are given at "Notes on Installing Angstrom via Narcissus"[14].
4. We compiled Boost libraries in BBx because we need newest version for our project.
5. We connected the camera and executed the test code on BBx.

After following these steps, the development environment is ready.

4.2.2 Programming Process

Training part of Viola-Jones Object Detection Algorithm requires a lot of space and a lot of time. Because of that we have decided to use the tools provided by OpenCV. These tools are CascadeObjectDetector class and haar cascade features XML file. Furthermore, we are supposed to send C++ objects to the server. Serializing and deserializing operation of any object can be tiring. Therefore, we have used serialization package of Boost library described in [15].

C++ is an object-oriented language and we take the advantage of this. There are 2 classes and a main function in our project.

The first class is called Detector class which encapsulates 1 attribute and 3 methods. The attribute is a CascadeClasifier instance. CascadeClasifier is an object clasifier which uses extended version of Viola-Jones algorithm for object detection in OpenCV. The methods in Detector class are preProcess, detectFaces and postProcess. PreProcess method applies some preprocessing

operations on the image. DetectFaces method takes image as a parameter and detects faces in that image with the help of CascadeClasifier instance. PostPocess method extracts the faces from image and returns the normalized face images.

The second class is Recognizer class which contains base and extended implementation of eigenface method described at Section 2.2. It also encapsulates the trained value of pattern vectors, eigenfaces, eigenvalues and thresholds. The Recognizer class has 2 methods for training. One method is the implementation of default eigenface algorithm and the other one is the extended version of that algorithm.

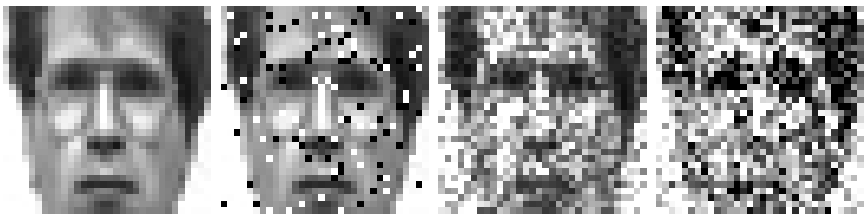
The main function is where the code starts and ends the execution. The process of main function is as follows:

- First, there is a socket connection between client and server. In order to open a socket connection, we used the functions of the C++ Standard Socket Library.
- After the establishment of connection, client creates and sends a string which contains the location information.
- The serialized Recognizer object is read as a response of previous message. Then, client creates a Detector object.
- After these initialization steps, client captures frames from the usb camera. This operation is realized by cvQueryFrame function of OpenCV.
- Our recognition algorithm and viola jones object detection algorithm works with gray scale images. Therefore, client converts the RGB images to gray scale images.
- Detector object take frames, detects faces in that frames and stores their coordinates to a vector as rectangle objects.
- For each face coordinate, Recognizer's predict function is called. If prediction is successful, student id is added to attendance vector.
- Finally, the attendance vector is converted to a string with serialization library of Boost and sent to the server.

5 Experiment Results

5.1 Preliminary Experiment Results

These experiments are realized on Matlab to measure the success of the algorithms. We have used the Yale database in our experiments. There are 165 face images, belonging to 15 people, in this database. First, we used MATLAB's object detector, which uses Viola-Jones object detection algorithm, to extract faces from entire image. Then, we divided this database into two different sets. We used 30% of these images as our test set. Then, we created noisy and rotated images from that test set. The sets of noisy images are composed of salt & pepper noise($d = 0.1$), speckle noise($\sigma = 0.1$) and Gaussian($\mu = 0, \sigma = 0.1$) noise. The examples of images in test sets are shown in Figure 14. Afterwards, we trained system with our training set. We have calculated two thresholds as explained in Section 3. The results of face recognition experiments with the proposed system are given in Table 1.



(a) Without Noise (b) Salt & Pepper Noise (c) Speckle Noise (d) Gaussian Noise

Figure 14: Example of images from different test sets.

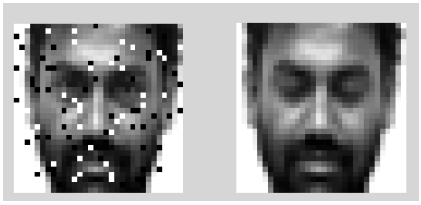
Recognition Rate	All Eigenfaces	90% of eigenfaces	90% of eigenfaces without the highest 2.
Normal Test Set(%)	69.05	69.05	**80.95**
Rotate with 3°(%)	61.90	57.14	**78.57**
Rotate with 5°(%)	50	50	**66.67**
Salt&Pepper Noise(%)	66.67	64.29	**73.18**
Speckle Noise(%)	54.76	54.76	**76.19**
Gaussian Noise(%)	50	50	**71.46**

Table 1: Successful Recognition Rate

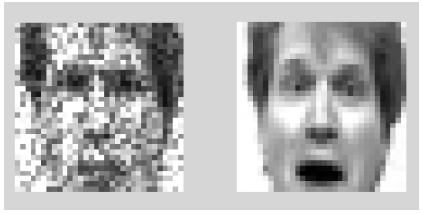
It can be seen from Figure 15a even if the visual quality of the images are poor, algorithm can correctly classifies the faces because the correlation between the pixels are still not corrupted much. However, success rate of the recognition process is still affected negatively. The example face images of the wrong recognition can be seen in Figure 16.

(a) Normal

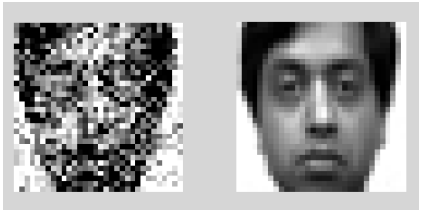
(b) 5° Rotated



(c) Salt & Pepper Noise

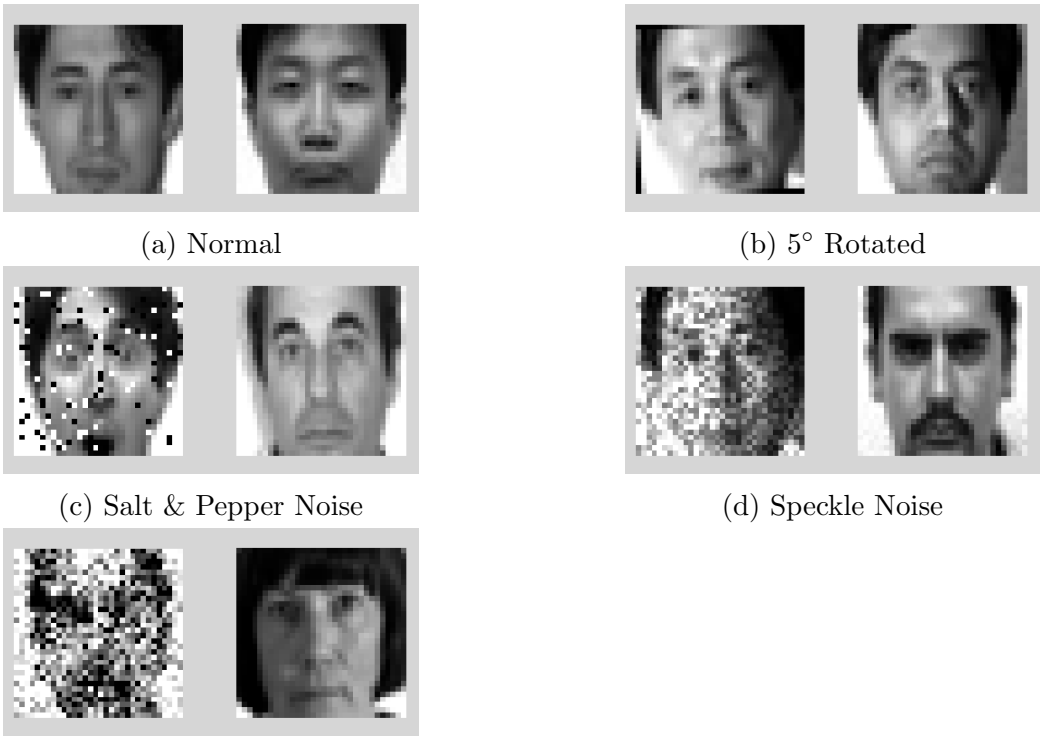


(d) Speckle Noise



(e) Gaussian Noise

Figure 15: Correct Examples of Recognition



(a) Normal

(b) 5° Rotated

(c) Salt & Pepper Noise

(d) Speckle Noise

(e) Gaussian Noise

Figure 16: Wrong Examples of Recognition

5.2 Experimental Results on BeagleBoard-xM

The aim of the experiments described in this section is to show how the ratio between training and test sets, and our extensions in eigenface algorithm affect the recognition and detection results in terms of speed and accuracy. In order to examine this, we have defined 4 groups for training to test ratio which are $50\%, 60\%, 70\%, 80\%$. In addition to that we created 10 different training and test sets for each train to test ratio randomly. In other words, there exists 400 training sets and 400 test sets.

The results shown in this section were calculated on BeagleBoard-xM.

Train/Test(%)	Accuracy(%)	Detection(sec)	Recognition(sec)
50% – *Base*	74 / 64.8	0.9134 / 0.9051	0.0341 / 0.0327
50% – *Extended*	77 / 71.1	0.7831 / 0.7752	0.0394 / 0.0304
60% – *Base*	71 / 64.4	0.9130 / 0.9001	0.0480 / 0.0454
60% – *Extended*	76 / 71.9	0.7762 / 0.7741	0.0439 / 0.0423
70% – *Base*	73 / 66.7	0.9280 / 0.9037	0.0600 / 0.0588
70% – *Extended*	85 / 77.8	0.7776 / 0.7713	0.0561 / 0.0556
80% – *Base*	85 / 70.6	0.9332 / 0.8977	0.0717 / 0.0714
80% – *Extended*	89 / 78.9	0.7767 / 0.7680	0.0703 / 0.0686

*(best/mean)

Table 2: Experiment Results

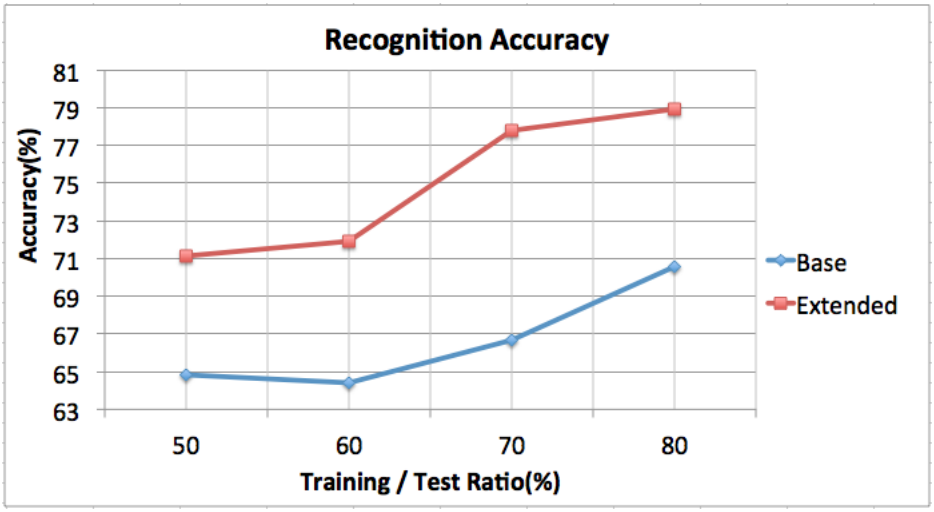


Figure 17: Recognition Results

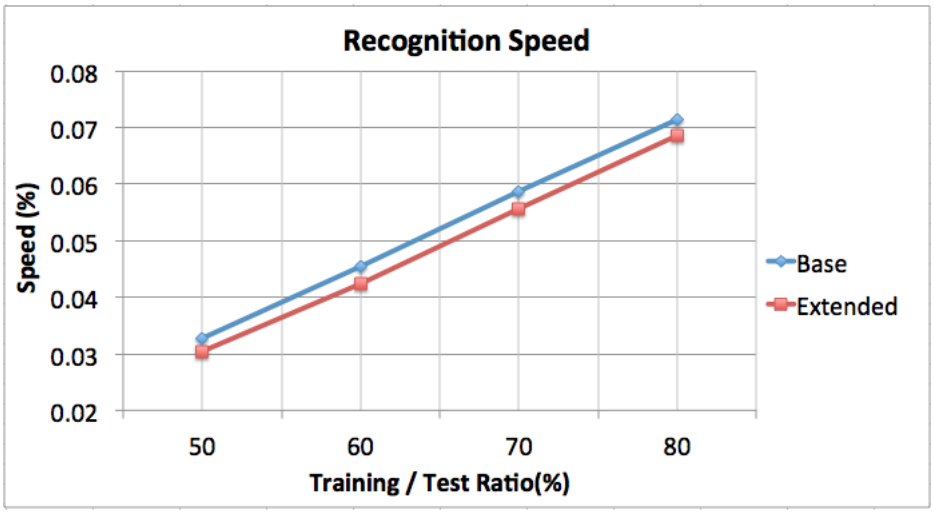


Figure 18: Recognition Results

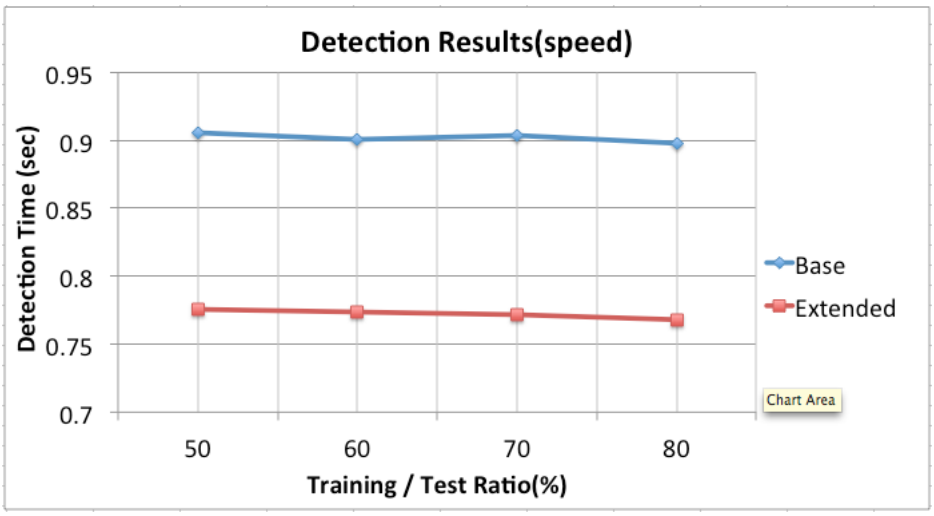


Figure 19: Detection Results

6 Conclusion

We have managed to run Viola-Jones object detection algorithm and eigenface face recognition method on different development environments. These environments are MATLAB and C++ with OpenCV and Boost libraries. We analyzed these algorithms in terms of their performance. We have modified some parts of the algorithms to boost their performances. With these modifications, we got approximately 8% more recognition rate. We realized that eliminating some principal components improves the success rate a lot. Despite these improvements, eigenface method does not perform well in different illumination conditions. Because of that we tried to equalize histogram in preprocessing part. However, it did not improve the recognition rate. The reason is that histogram equalization is not a linear transformation.

When we look at the embedded part of our project, we can execute complex operations on BeagleBoad-xM. Since there is an operating system on BeagleBoard-xM, it is easy to perform network operations and communicate with peripheral devices.

In conclusion, face recognition is still a hard problem especially on embedded hardware. We hope that our project will enlighten the future projects on this field.

References

[1] L. Sirovich and M.Kirby, Low-dimensional procedure for the characterization of human faces. In *Journal of the Optical Society of America A, 4(3), 519-524,* 1987.

[2] M. Turk and A. Pentland, Eigenfaces for Recognition. In *J. Cognitive Neuroscience, vol.3, no.1,* 1991.

[3] P. Viola and M. J. Jones, Robust real-time object detection. In *Proc. of IEEE Workshop on Statistical and Computational Theories of Vision,* 2001.

[4] R. Benveniste, B. Sırmaçek, C. Ünsalan 21.07.2010, A Quick Start To Texas Instruments TMS 320C6713 DSK. *http://www.ti.com/ww/eu/university/Yeditepe_C6713_DSK_LAB_MANUAL_and_4.pdf*

[5] Yale Face Image Database, *http://cvc.yale.edu/projects/yalefaces/yalefaces.html*

[6] MIT cbcl Face Image Database, *http://cbcl.mit.edu/software-datasets/FaceData2.html*

[7] M. Barr and A. Massa, Programming Embedded Systems: With C and GNU Development Tools, Second Edition, *O'REILLY*

[8] Netty Network Library, *http://netty.io/*

[9] High Performance Inter-Thread Messaging Library, *http://lmax-exchange.github.io/disruptor/*

[10] The Fast Java JDBC Connection Pool Library, *http://jolbox.com*

[11] Open Computer Vision Library, *http://www.opencv.org*

[12] Boost C++ Library, *http://www.boost.org*

[13] Narcissus - Online image builder for the angstrom distribution, *http://narcissus.angstrom-distribution.org*

[14] ECE497 Notes on Installing Angstrom via Narcissus, *http://elinux.org/ECE497_Lab01_Installing_Angstrom_via_Narcissus*

[15] Serialization of cv::Mat objects using Boost, *http://cheind.wordpress.com/2011/12/06/serialization-of-cvmat-objects-using-boost*

7 Appendix

A

TMS320C6713 DSP Starter Kit (DSK) is developed by Texas Instruments. It supports both fixed and floating point functionality.

- 225 MHz device delivering up to 1800 million instructions per second (MIPs) and 1350 MFLOPS
- Embedded JTAG support via USB
- High-quality 24-bit stereo codec
- Four 3.5mm audio jacks for microphone, line in, speaker, line out
- 512K words of flash 16 MB SDRAM
- Expansion port connector for plug-in modules
- On-board standard IEEE JTAG interface
- +5V universal power supply

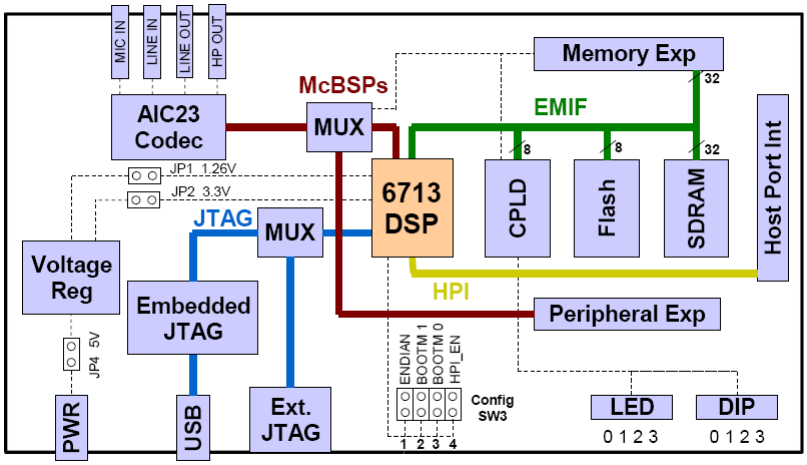


Figure 20: Architecure of C6713

8 Appendix

B

BeagleBoard-xM is developed by Texas Instruments.

- Processor: AM37x 1GHz ARM Cortex-A8 compatible
- More than 2,000 Dhrystone MIPS
- Up to 20 million polygons/second graphics
- HD video capable C64+TMDSP core
- 512 MB LPDDR RAM
- 2D/3D graphics accelerator
- 4 USB 2.0 ports
- MMC/SD connector
- DVI-D port
- S-Video port
- USB mini AB connector
- Ethernet
- Angstrom Linux

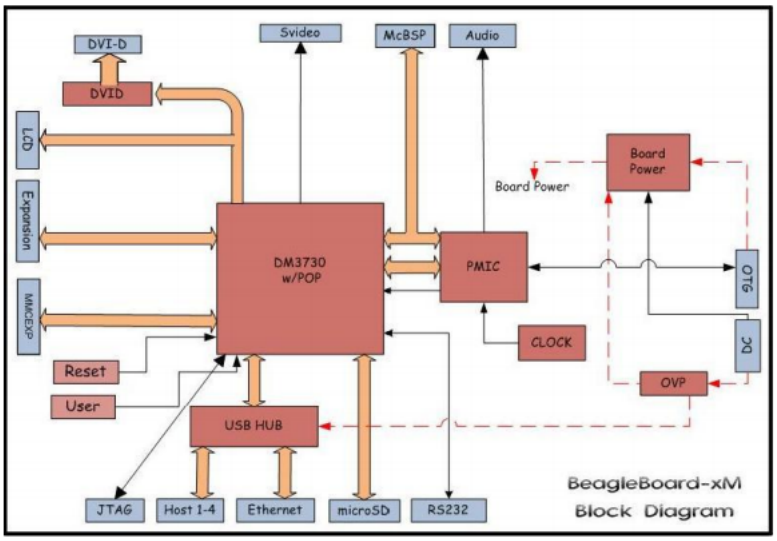


Figure 18. BeagleBoard-xM High Level Block Diagram

Figure 21: Architecure of BeagleBoard-xM