BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF MICROELECTRONICS

KATHOLIEKE HOGESCHOOL BRUGGE–OOSTENDE

DEVELOPMENT OF ALGORITHMS FOR DIGITAL REAL TIME IMAGE PROCESSING ON A DSP PROCESSOR (FACE DETECTION AND FACE RECOGNITION)

MASTER'S THESIS

AUTHOR Bc. PETER KNAPO

SUPERVISOR ir. JURGEN BAERT (MSc), KHBO Belgium

BRNO, OOSTENDE 2009

LICENČNÍ SMLOUVA POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan

Jméno a příjmení: Peter KNAPO

Bytem: Papradno 286, 018 13 Papradno, Slovenská Republika

Narozen/a (datum a místo): 18.5.1984 v Považskej Bystrici

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

Prof. Ing. Radimír Vrba, CSc.

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   □ diplomová práce
   (dále jen VŠKP nebo dílo)

Název VŠKP:	Vývoj algoritmů pro digitální zpracování obrazu v reálním čase v DSP procesoru
Vedoucí/ školitel VŠKP:	ir. Jurgen Baert (MSc)
Ústav:	KHBO Belgium
Datum obhajoby VŠKP:	20.8.2009

VŠKP odevzdal autor nabyvateli v*:

□ tištěné formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   □ ihned po uzavření této smlouvy
   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: ……………………………………

……………………………………… ………………………………………
Nabyvatel Autor

ABSTRACT

Face recognition is a complex process that aims to recognize human faces in images or video sequences. Applications include surveillance and identification system, but face recognition is also invaluable in the research of computer vision and artificial intelligence. Face recognition systems are often based on either image analysis or neural networks. This work implements an algorithm based around the use of so-called eigenfaces. Eigenfaces are the result of a form of Principal Component Analysis (PCA), which extracts important facial features from the original image and is based on solving a linear matrix equation of the covariance matrix, eigenvalues and eigenvectors. A face that is to be recognized is thus projected onto the eigenspace; the results of that operation can be interpreted as the comparison of this face with an existing database of known faces. Before executing the actual recognition algorithm, faces need to be located inside the image and prepared (by doing normalization, lighting compensation and noise removal). Many algorithms exist, but this work uses a color based face detection algorithm, which is both fast and sufficient for this application. The face detection and recognition algorithms are implemented on a Blackfin ADSP-BF561 DSP processor from Analog Devices.

KEYWORDS

Face detection, Face recognition, Skin color based face detection algorithm, Face recognition based on Eigenfaces, PCA, Principal Component Analysis, Eigenfaces, Eigenvector, Eigenvalue, DSP processor, C implementation, ADSP-BF561, EZ-KIT Lite, Digital image processing.

ABSTRAKT

Rozpoznávanie tvárí je komplexný proces, ktorého hlavným cieľom je rozpoznanie ľudskej tváre v obrázku alebo vo video sekvencii. Najčastejšími aplikáciami sú sledovacie a identifikačné systémy. Taktiež je rozpoznávanie tvárí dôležité vo výskume počítačového videnia a umelej inteligencií. Systémy rozpoznávania tvárí sú často založené na analýze obrazu alebo na neurónových sieťach. Táto práca sa zaoberá implementáciou algoritmu založeného na takzvaných „Eigenfaces“ tvárach. „Eigenfaces“ tváre sú výsledkom Analýzy hlavných komponent (Principal Component Analysis - PCA), ktorá extrahuje najdôležitejšie tvárové črty z originálneho obrázku. Táto metóda je založená na riešení lineárnej maticovej rovnice, kde zo známej kovariančnej matice sa počítajú takzvané „eigenvalues“ a „eigenvectors“, v preklade vlastné hodnoty a vlastné vektory. Tvár, ktorá má byť rozpoznaná, sa premietne do takzvaného „eigenspace“ (priestor vlastných hodnôt). Vlastné rozpoznanie je na základe porovnania takýchto tvárí s existujúcou databázou tvárí, ktorá je premietnutá do rovnakého „eigenspace“. Pred procesom rozpoznávania tvárí, musí byť tvár lokalizovaná v obrázku a upravená (normalizácia, kompenzácia svetelných podmienok a odstránenie šumu). Existuje mnoho algoritmov na lokalizáciu tváre, ale v tejto práci je použitý algoritmus lokalizácie tváre na základe farby ľudskej pokožky, ktorý je rýchly a postačujúci pre túto aplikáciu. Algoritmy rozpoznávania tváre a lokalizácie tváre sú implementované do DSP procesoru Blackfin ADSP-BF561 od Analog Devices.

KLÍČOVÁ SLOVA

Lokalizace obličeje, Rozpoznávání tváří, Algoritmus lokalizace obličeje na základě barvy lidské kůže, Rozpoznávání tváří s využitím Eigenfaces, PCA, Principal Component Analysis, Eigenfaces, Eigenvector, Eigenvalue, DSP procesor, Implementace v C, ADSP-BF561, EZ-KIT Lite, Číslicové zpracování obrazu.

KNAPO, P. Vývoj algoritmů pro digitální zpracování obrazu v reálním čase v DSP procesoru. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 93 s. Vedoucí diplomové práce ir. Jurgen Baert (MSc), KHBO Belgium.

PREHLÁSENIE

Prehlasujem, že som diplomovú prácu na tému "Vývoj algoritmů pro digitální zpracování obrazu v reálním čase v DSP procesoru" vypracoval samostatne pod vedením vedúceho práce, s použitím literárnych prameňov a publikácií, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto práce som neporušil autorské práva tretích osôb a hlavne, že som nezasiahol nepovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomí následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia § 152 trestného zákona č. 140/1961 Sb.

V Brně dňa 10.08.2009

podpis

ACKNOWLEDGEMENTS

I would like to thank my project leader Jurgen Baert (Katholieke Hogeschool Brugge–Oostende, campus Oostende, Belgium) for managing of this project and giving me help and information. Same thank belongs to other teachers from this university: Renaat De Craemer, Joan Peuteman, Jean-Jacques Vandenbussche, Franky Loret, Sammy Verslype, Gwendolyn Rogge, for helping during my stay in Oostende and for giving me an opportunity doing my diploma project at this university.

Also, I would like to thank Vladislav Musil and Edita Hejátková from Department of Microelectronics, Brno University of Technology, for helping with organization of this project.

TABLE OF CONTENTS

FIGURES

TABLES

Preface

This text (master's thesis) represents formulation of my diploma (master's) project. I was working on this project at *Katholieke Hogeschool Brugge–Oostende / Faculty of Industrial Engineering Sciences and Technology* in Belgium as my international exchange stay (LLP - Erasmus) at this university for 5 month.

Project deals with implementation of some algorithms from image processing domain and other domains into the DSP processor Blackfin® ADSP-BF561 from Analog Devices, Inc. with using a development board EZ-KIT Lite® ADSP-BF561 from the same company.

The implemented algorithms are: Frame processing and image modification, Face detection, Face recognition, Communication with PC via UART, Video capturing from camera, and others. Also, I have created a Windows application which communicates with DSP processor via UART.

Some terms used in this thesis are not fully grammatically correct, because are rooted from English researches and publications.

This work is divided into the several chapters:

The first chapter is Introduction, where I will describe basic knowledge about digital image processing. Also, I will describe specific color models and video format for video streaming. This information is good to know to understand this project.

In the second chapter are described development tools, which were used for designing this project, such as DSP processor, development board and development environment.

The third chapter describes Skin Color based face detection algorithm. Firstly, I will describes face detection – "what is it" and existing methods and approaches. Then, will be description of proposed face detection algorithm and its implementation into DSP processor using C/C++. The last section of this chapter is experimental results of this face detection algorithm.

The fourth chapter describes face recognition algorithm based on Eigenfaces (Principal component analysis). In the beginning of this chapter will be introduction to face recognition - existing methods and approaches. Then, will be described this face recognition algorithm and then its implementation into DSP processor using C/C++. Last section of this chapter is experimental results of proposed face recognition algorithm.

If you have any comment concerning this project, please contact me on e-mail: peter.knapo@gmail.com .

Author

1 Introduction

This chapter gives the basic knowledge and overview of digital image processing and its parts, and describes specific color models and video format for video streaming. This information are needful to better understand problems aimed in this diploma project.

1.1 Digital image processing

An image may be defined as a two-dimensional function, *f(x, y)*, where x and y are spatial (plane) coordinates, and the amplitude of *f* at any pair of coordinates *(x, y)* is called the intensity or gray level of the image at that point. When *x*, *y*, and the amplitude values of *f* are all finite, discrete quantities, we call the image a digital image. The field of digital image processing refers to processing digital images by means of a digital computer. Note, that a digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are referred to as picture elements, image elements, and pixels. Pixel is the term most widely used to denote the elements of a digital image. (cit. [1])

1.1.1 Fundamentals of digital image processing

(cit. [2])

We are in the middle of a visually enchanting world, which manifests itself with a variety of forms and shapes, colors and textures, motion and tranquility. The human perception has the capability to acquire, integrate, arid interpret all this abundant visual information around us. It is challenging to impart such capabilities to a machine in order to interpret the visual information embedded in still images, graphics, and video or moving images in our sensory world. It is thus important to understand the techniques of storage, processing, transmission, recognition, and finally interpretation of such visual scenes.

The first step towards designing an image analysis system is digital image acquisition using sensors in optical or thermal wavelengths. A two dimensional image that is recorded by these sensors is the mapping of the three-dimensional visual world. The captured two dimensional signals are sampled and quantized to yield digital images.

Sometimes we receive noisy images that are degraded by some degrading mechanism. One common source of image degradation is the optical lens system in a digital camera that acquires the visual information. If the camera is not appropriately focused then we get blurred images. Here the blurring mechanism is the defocused camera. Very often one may come across images of outdoor scenes that were procured in a foggy environment. Thus any outdoor scene captured on a foggy winter morning could invariably result into a blurred image. In this case the degradation is due to the fog and mist in the atmosphere, and this type of degradation is known as atmospheric degradation. In some other cases there may be a relative motion between the object and the camera. Thus if the camera is given an impulsive displacement during the image capturing interval while the object is static, the resulting image will invariably be blurred and noisy. In some of the above cases, we need appropriate techniques of refining the images so that the resultant images are of better visual quality, free from aberrations and noises. Image enhancement, filtering, and restoration have been some of the important applications of image processing.

Segmentation is the process that subdivides an image into a number of uniformly homogeneous regions. Each homogeneous region is a constituent part or object in the entire scene. In other words, segmentation of an image is defined by a set of regions that are connected and non overlapping, so that each pixel in a segment in the image acquires a unique region label that indicates the region it belongs to. Segmentation is one of the most important elements in automated image analysis, mainly because at this step the objects or other entities of interest are extracted from an image for subsequent processing, such as description and recognition. For example, in case of an aerial image containing the ocean and land, the problem is to segment the image initially into two parts-land segment and water body or ocean segment. Thereafter the objects on the land part of the scene need to be appropriately segmented and subsequently classified.

After extracting each segment; the next task is to extract a set of meaningful features such as texture, color, and shape. These are important measurable entities which give measures of various properties of image segments. Some of the texture properties are coarseness, smoothness, regularity, etc., while the common shape descriptors are length, breadth, aspect ratio, area, location, perimeter, compactness, etc. Each segmented region in a scene may be characterized by a set of such features.

Finally based on the set of these extracted features, each segmented object is classified to one of a set of meaningful classes. In a digital image of ocean, these classes may be ships or small boats or even naval vessels and a large class of water body. The problems of scene segmentation and object classification are two integrated areas of studies in machine vision. Expert systems, semantic networks, and neural network-based systems have been found to perform such higher-level vision tasks quite efficiently.

1.2 Color models and transformation

A number of color spaces or color models have been suggested and each one of them has a specific color coordinate system and each point in the color space represents only one specific color. Each color model may be useful for specific applications. There are a number of such color spaces like RGB, CIELAB, YCbCr, CMYK, HSV, HIS, or LUV, etc.

1.2.1 RGB color model

Typical color images, particularly those generated by a digital imaging system, are represented as red, green, blue, and are normally called RGB images. They are useful for color monitors, and video cameras.

In RGB model, each color appears in its primary spectral components of red, green, and blue. This model is based on a Cartesian coordinate system. The color subspace of interest is the cube shown in Figure 1, where RGB values are at three corners; cyan, magenta and yellow are at three other corners; black is at the origin; and white is at the corner farthest from the origin. In this model, the gray scale (points of equal RGB values) extends from black to white along the joining these two points. The different colors in this model are points on or inside the cube, and are defined by vectors extending from origin. For convenience, the assumption is that all color values have been normalized. That is, all values of R, G, and B are assumed to be in the range [0, 1]. (cit. [1])

An RGB color image, represented by 8 bits of R, G, and B pixels has 256^3 or 16 777 216 colors.

Transformation RGB -> Grayscale level [3]:

$$GrayScale = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \qquad (1)$$

Where R, G, B, and GrayScale are normalized in the range [0, 1].

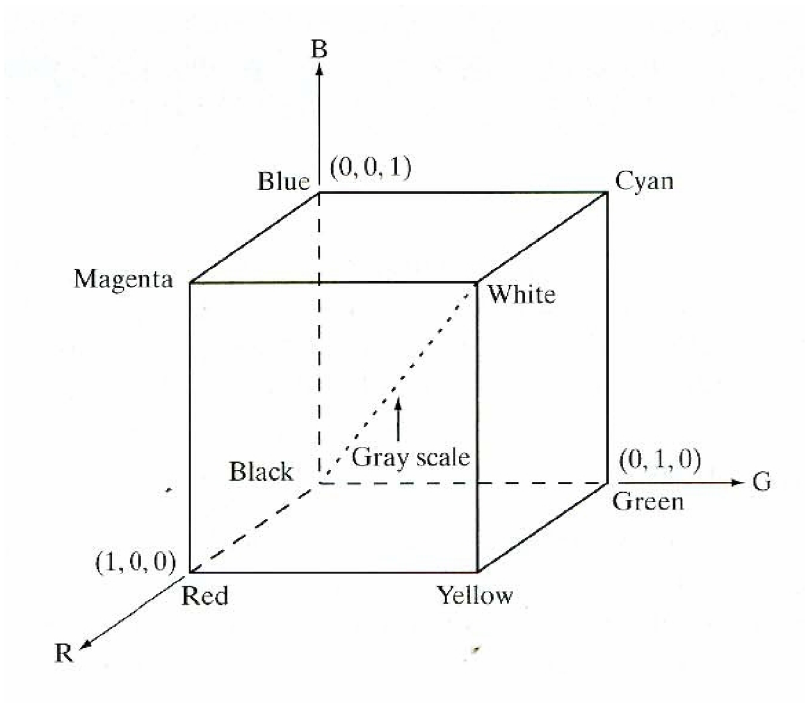


Figure 1: *RGB color cube (normalized RGB model).*

1.2.2 YCbCr color model

In this color model, Y is the luminous component while Cb and Cr provide the color information. The color information is stored as two color difference components Cb and Cr.

Transformation RGB -> YCbCr [2]:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.00 \\ 112.00 & -93.786 & -18.214 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (2)$$

Where R, G, B; Y, Cb, and CR are normalized in the range [0, 255].

The YCbCr color model is used in ITU-R 656 video mode for coding colors in digital video stream (NTSC or PAL format).

1.3 ITU-R 656 video mode for digital video streaming

The ITU-R 656 video mode is used for many digital video streams such as video in/out from monitor, TV or portable camera. The ITU-R 656 is supported many of DSP processors, for example, DSP processor Blackfin ADSP-BF561 from Analog Devices which is used for this project.

(cit. [4])

According to the ITU-R 656 recommendation (formerly known as CCIR-656), a digital video stream has the characteristics shown in Figure 2, and Figure 3 for 525/60 (NTSC) and 625/50 (PAL) systems.

In this mode, the Horizontal (H), Vertical (V), and Field (F) signals are sent as an embedded part of the video datastream in a series of bytes that form a control word. The Start of Active Video (SAV) and End of Active Video (EAV) signals indicate the beginning and end of data elements to read in on each line. SAV occurs on a 1-to-0 transition of H, and EAV begins on a 0-to-1 transition of H. An entire field of video is comprised of Active Video + Horizontal Blanking (the space between an EAV and SAV code) and Vertical Blanking (the space where V = 1). A field of video commences on a transition of the F bit. The "odd field" is denoted by a value of F = 0, whereas F = 1 denotes an even field. Progressive video makes no distinction between Field 1 and Field 2, whereas interlaced video requires each field to be handled uniquely, because alternate rows of each field combine to create the actual video image.

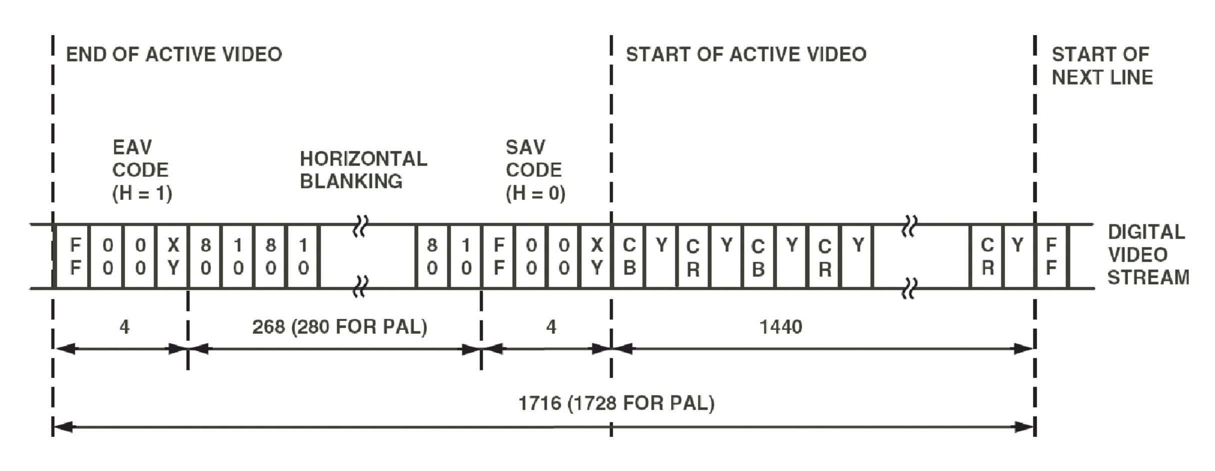
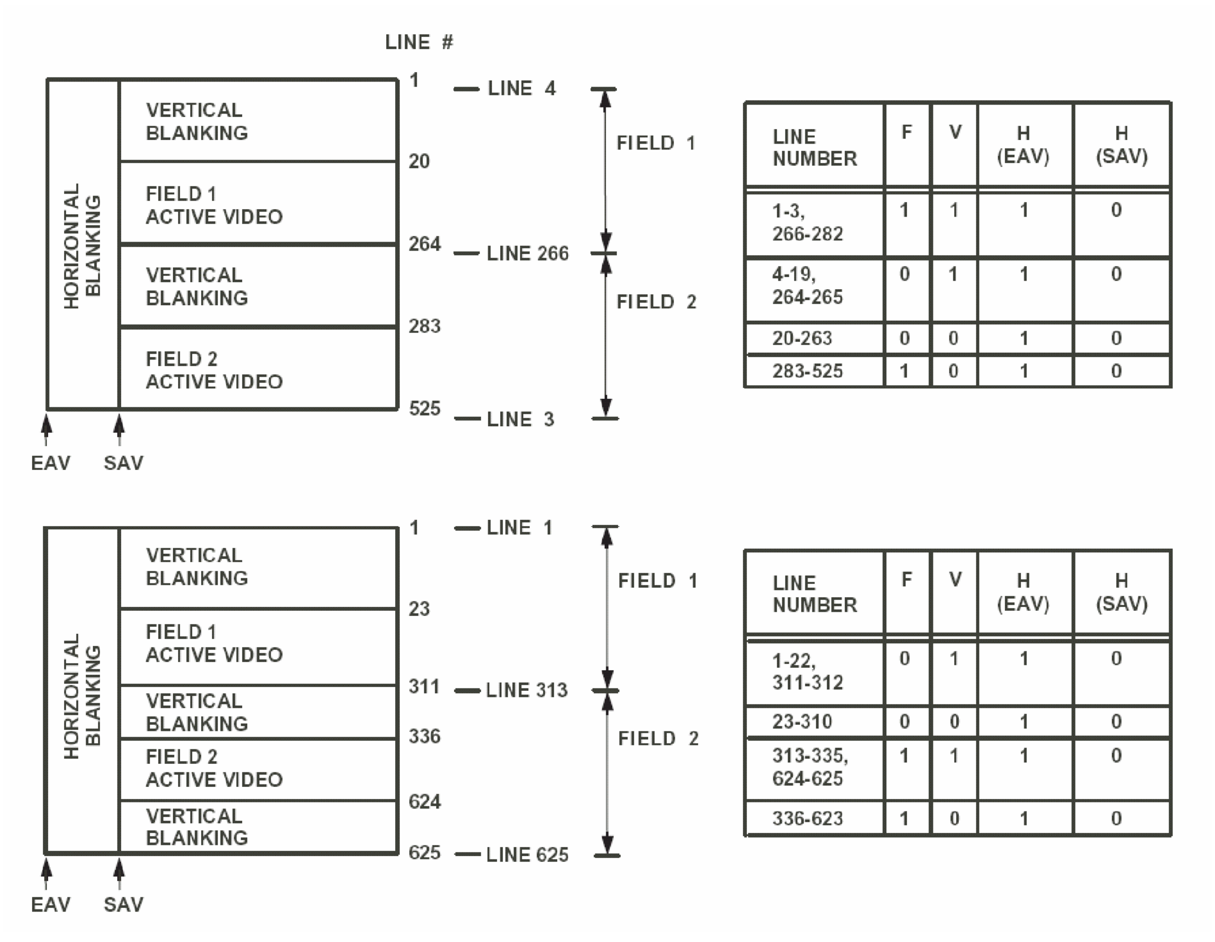


Figure 2: *ITU-R 656 8-Bit Parallel Data Stream for NTSC (PAL) Systems.*

In many applications, video streams are other than the standard NTSC/PAL formats (for example, CIF, QCIF) can be employed. Because of this, the processor interface is flexible enough to accommodate different row and field lengths. In general, as long as the incoming video has the proper EAV/SAV codes, the PPI can read it in. In other words, a CIF image could be formatted to be “656-compliant,” where EAV and SAV values define the range of the image for each line, and the V and F codes can be used to delimit fields and frames.



LINE NUMBER	F	V	H (EAV)	H (SAV)
1-3, 266-282	1	1	1	0
4-19, 264-265	0	1	1	0
20-263	0	0	1	0
283-525	1	0	1	0

LINE NUMBER	F	V	H (EAV)	H (SAV)
1-22, 311-312	0	1	1	0
23-310	0	0	1	0
313-335, 624-625	1	1	1	0
336-623	1	0	1	0

Figure 3: *Typical Video Frame Partitioning for NTSC/PAL Systems for ITU-R BT.656-4.*

<u>NTSC definitions:</u>

§ Resoultion: 720x480

§ 525/60 video system

<u>PAL definitions:</u>

§ Resoultion: 720x576

§ 625/50 video system

In ITU-R 656 video mode there are 2 types of YCbCr sampling [5]:

§ 4:4:4 (full sampling, which means that every sample has Y, Cb, and Cr component), shown in Figure 4.

§ 4:2:2 (used for standard ITU-R 656 video mode; not full sample, which means that Y, Cb, and Cr component has only every second sample. Others samples have only Y component), shown in Figure 5.

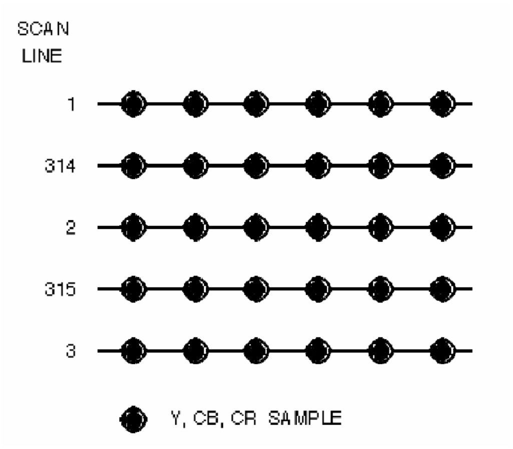


Figure 4: *4:4:4 YCbCr sampling structure for a 625-line interlaced system.*

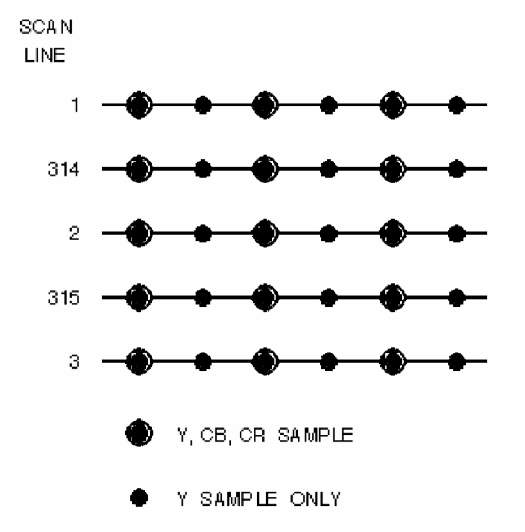


Figure 5: *4:2:2 YCbCr sampling structure for a 625-line interlaced system.*

2 Development tools

This chapter describes development board EZ-KIT Lite® ADSP-BF561 with digital signal processor Blackfin® ADSP-BF561 and development environment VisualDSP++ from Analog Devices, Inc. The DSP algorithms are implemented into this processor using this environment.

2.1 DSP processor Blackfin® ADSP-BF561

(cit. [6])

The ADSP-BF561 processor is a high performance member of the Blackfin® family of products targeting a variety of multimedia, industrial, and telecommunications applications. At the heart of this device are two independent Analog Devices Blackfin processors. These Blackfin processors combine a dual-MAC state-of-the-art signal processing engine, the advantage of a clean, orthogonal RISC-like microprocessor instruction set, and single instruction, multiple data (SIMD) multimedia capabilities in a single instruction set architecture.

<u>FEATURES:</u>

§ Dual symmetric 600 MHz high performance Blackfin cores

§ 328K bytes of on-chip memory

§ Each Blackfin core includes:

  § Two 16-bit MACs, two 40-bit ALUs, four 8-bit video ALUs, 40-bit shifter

  § RISC-like register and instruction model for ease of programming and compiler-friendly support

  § Advanced debug, trace, and performance monitoring

§ 0.8 V to 1.35 V core VDD with on-chip voltage regulator

§ 2.5 V and 3.3 V compliant I/O

§ 256-ball CSP_BGA (2 sizes) and 297-ball PBGA package options

The ADSP-BF561 processor has 328K bytes of on-chip memory.

Each Blackfin core includes:

§ 16K bytes of instruction SRAM/cache

§ 16K bytes of instruction SRAM

§ 32K bytes of data SRAM/cache

§ 32K bytes of data SRAM

§ 4K bytes of scratchpad SRAM

Additional on-chip memory peripherals include:

§ 128K bytes of low latency on-chip L2 SRAM

§ Four-channel internal memory DMA controller

§ External memory controller with glue-less support for SDRAM, mobile SDRAM, SRAM, and flash.

PERIPHERALS:

§ Dual 12-channel DMA controllers (supporting 24 peripheral DMAs)

§ 2 memory-to-memory DMAs

§ internal memory-to-memory DMAs and 1 internal memory DMA controller

§ 12 general-purpose 32-bit timers/counters with PWM capability

§ SPI-compatible port

§ UART with support for IrDA

§ Dual watchdog timers

§ Dual 32-bit core timers

§ 48 programmable flags (GPIO)

§ On-chip phase-locked loop capable of 0.5× to 64× frequency multiplication

§ parallel input/output peripheral interface (PPI0, PPI1) units supporting ITU-R 656 video and glue-less interface to analog front end ADCs

§ 2 dual channel, full duplex synchronous serial ports supporting (SPORT0, SPORT1) eight stereo I2S channels

Figure 6 shows functional block diagram of dual-core Blackfin® ADSP-BF561 processor and Figure 7 shows one core of this processor.

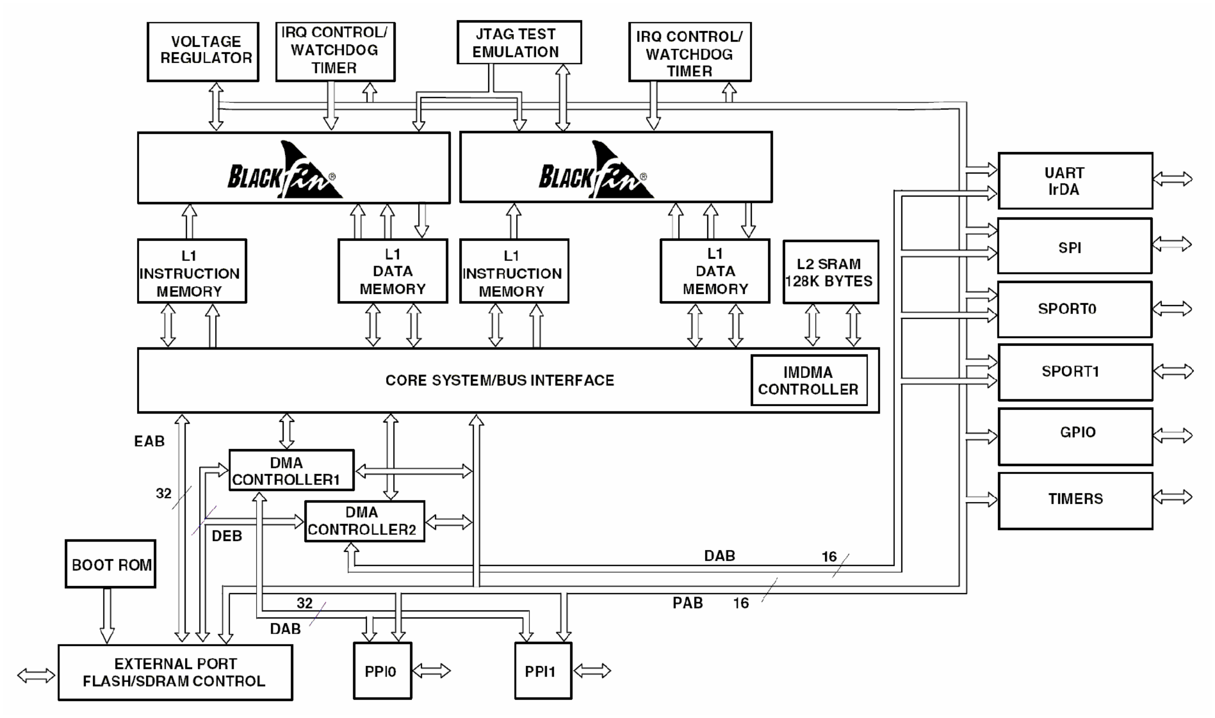


Figure 6: *Blackfin® ADSP-BF561 - Functional Block Diagram.*

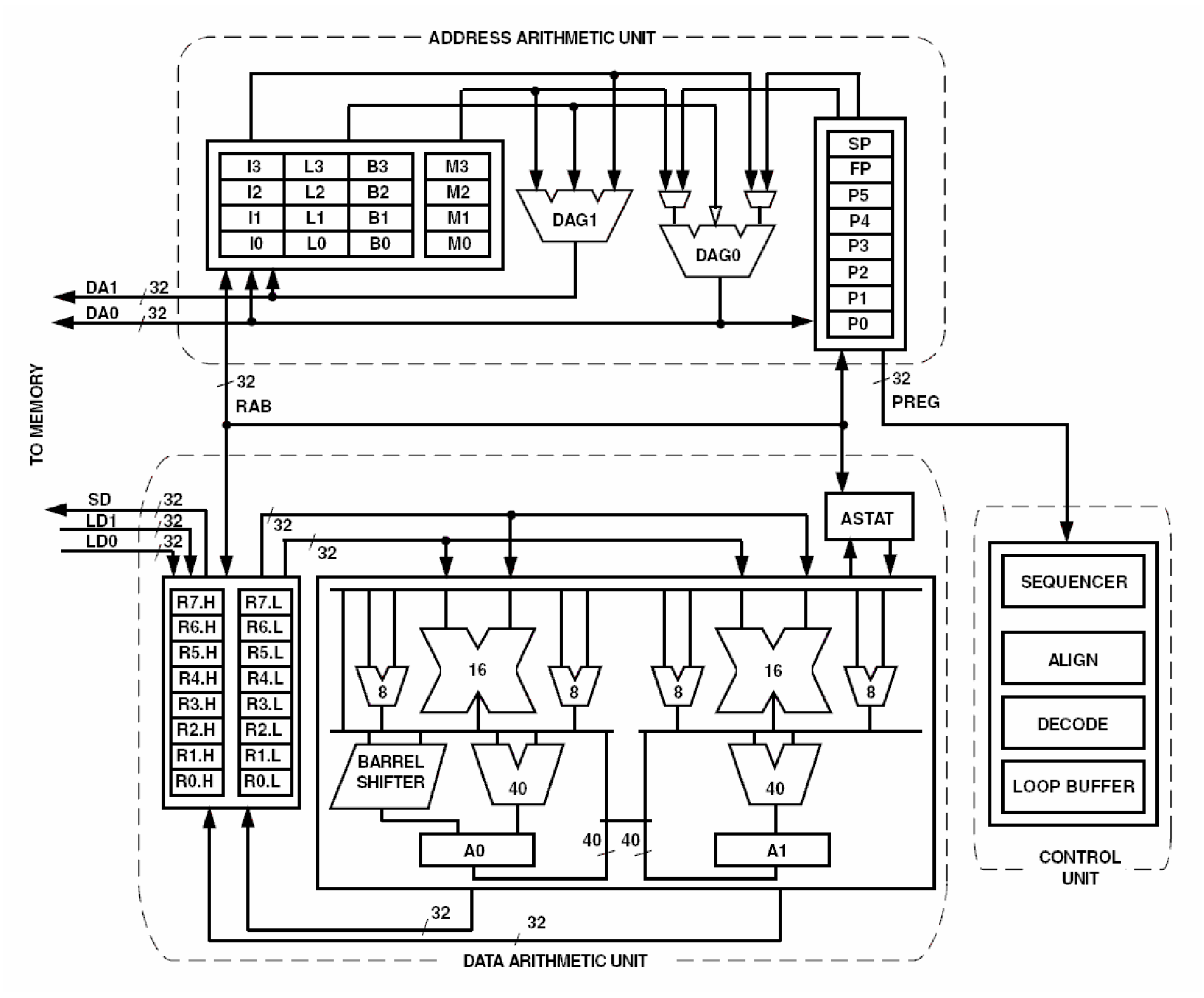


Figure 7: *Blackfin®ADSP-BF561 – One core.*

2.2 Development board EZ-KIT Lite® ADSP-BF561

(cit. [7])

This EZ-KIT Lite has been designed to demonstrate the capabilities of the ADSP-BF561 Blackfin processor. The processor has an IO voltage of 3.3V. The core voltage and the core clock rate can be set on the fly by the processor. The input clock is 30 MHz.

Figure 8 shows system architecture of Development board EZ-KIT Lite and Figure 9 shows picture of this board.

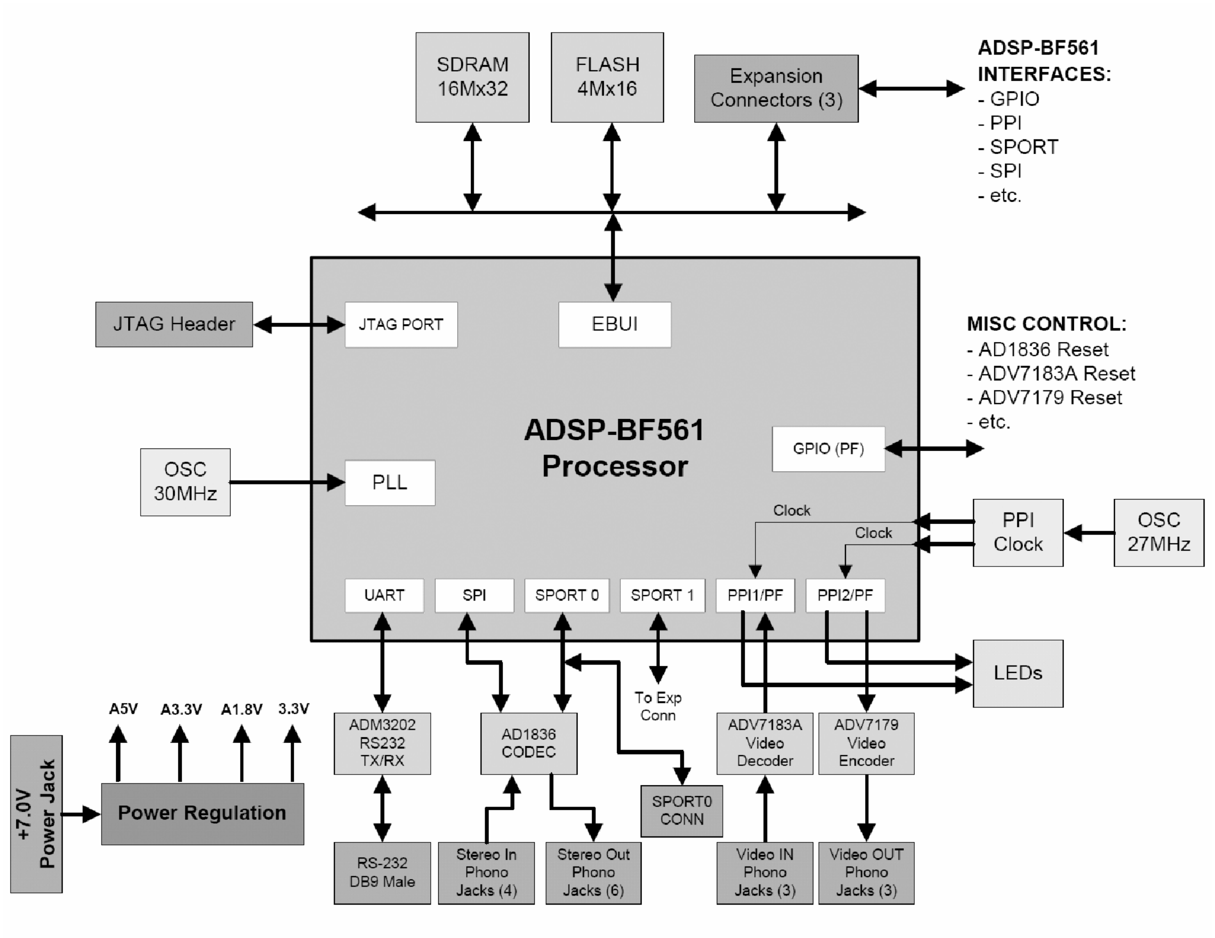


Figure 8: *Development board EZ-KIT Lite®ADSP-BF561 - System Architecture.*

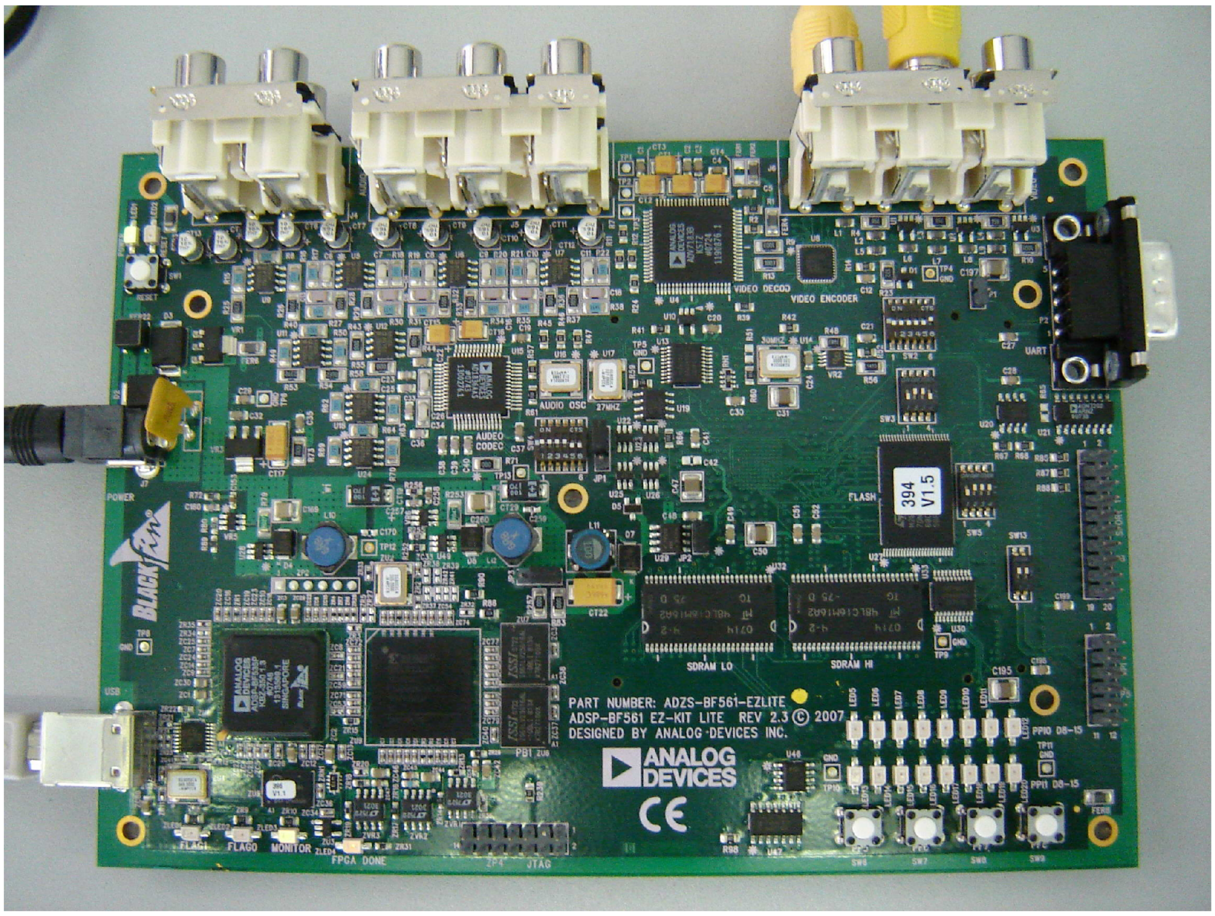


Figure 9: *Development board EZ-KIT Lite®ADSP-BF561 – Real picture.*

Figure in Appendix B1 shows Project diagram, where are demonstrated this development board and implemented algorithms.

THE BOARD FEATURES:

§ Analog Devices ADSP-BF561 Blackfin processor

  § 256-pin mini-BGA package

  § 30 MHz CLKIN oscillator

§ Synchronous dynamic random access memory (SDRAM)

  § 64 MB (16M x 16 bits x 2 chips)

§ Flash memory

  § 8 MB (4M x 16 bits)

- Analog audio interface
    - AD1836A – Analog Devices 96 kHz audio codec
    - 4 input RCA phono jacks (2 stereo channels)
    - 6 output RCA phono jacks (3 stereo channels)
- Analog video interface
    - ADV7183A video decoder w/ 3 input RCA phono jacks
    - ADV7179 video encoder w/ 3 output RCA phono jacks
- Universal asynchronous receiver/transmitter (UART)
    - ADM3202 RS-232 line driver/receiver
    - DB9 male connector
- LEDs
    - 20 LEDs: 1 power (green), 1 board reset (red), 1 USB (red), 16 general-purpose (amber), and 1 USB monitor (amber)
- Push buttons
    - 5 push buttons with debounce logic: 1 reset, 4 programmable flags
- Expansion interface
    - PPI0, PPI1, SPI, EBIU, Timers11-0, UART, programmable flags, SPORT0, SPORT1
- Other features
    - JTAG ICE 14-pin header

VIDEO INTERFACE:

The board supports video input and output applications. The ADV7179 video encoder provides up to three output channels of analog video, while the ADV7183A video decoder provides up to three input channels of analog video. The video encoder connects to the parallel peripheral interface 1 (PPI1), while the video decoder connects to the parallel peripheral interface 0 (PPI0).

2.3 Development environment VisualDSP++

(cit. [13])

VisualDSP++ is a development environment for DSP processors from Analog Devices, including Blackfin®ADSP-BF561.

<u>VisualDSP++ Features:</u>

VisualDSP++ includes all the tools needed to build and manage processor projects.

VisualDSP++ includes:

§ Integrated Development and Debugging Environment (IDDE) with VisualDSP++ Kernel (VDK) integration

§ C/C++ optimizing compiler with run-time library

§ Assembler and linker

§ Simulator software

§ Example programs

3 Skin Color based face detection algorithm

This chapter describes Face detection generally and implementation of Skin Color based face detection algorithm into DSP processor Blackfin®ADSP-BF561. In the end of this chapter are experimental results of implemented algorithm.

3.1 Introduction to Face detection

Face detection is algorithm to find and localize human face (or more faces) in picture or video frames. Main applications are identification and surveillance systems, localization face to help focus in cameras, pre-processing before image processing with faces like face recognition, and many of applications. Face detection is a necessary first step in all of the face processing systems and its performance can severely influence on the overall performance of recognition.

3.1.1 Existing methods

During the last decade a number of promising face detection algorithms have been developed and published. Detection algorithms can classify into four categories.

Face detection categories [8]:

§ Knowledge-based methods: These rule-based methods encode human knowledge of what constitutes a typical face. Usually, the rules capture the relationships between facial features. These methods are designed mainly for face localization.

§ Feature invariant approaches: These algorithms aim to find structural features that exist even when the pose, viewpoint, or lighting conditions vary, and then use these to locate faces. These methods are designed mainly for face localization. Approaches: Facial Features, Texture, Skin Color, Multiple Features.

§ Template matching methods: Several standard patterns of a face are stored to describe the face as a whole or the facial features separately. The correlations between an input image and the stored patterns are computed for detection. These methods have been used for both face localization and detection. Approaches: Predefines face templates, Deformable Templates.

§ Appearance-based methods: In contrast to template matching, the models (or templates) are learned from a set of training images which should capture the representative variability of facial appearance. These learned models are then used for detection. These methods are designed mainly for face detection. Approaches: Eigenface, Distribution-based, Neural Network, Support Vector Machine (SVM), Naïve Bayes Classifier, Hidden Markov Model (HMM), Information-Theoretical Approach.

In this work is used Skin Color approach to localize face, which is part of Feature invariant category.

Viola-Jones Face detection algorithm [9] and OpenCV library [10][11]:

For the present, it exist very effective algorithm for face detection – "Viola-Jones", which is very reliable and fast, simultaneously. The basic principle of the Viola-Jones algorithm is to scan a sub-window capable of detecting faces across a given input image. Detector is constructed using a so-called integral image and some simple rectangular features reminiscent of Haar wavelets. Viola-Jones uses a modified version of the AdaBoost algorithm. AdaBoost is a machine learning boosting algorithm capable of constructing a strong classifier through a weighted combination of weak classifiers. This algorithm is implemented into OpenCV library.

OpenCV is an Open Source Computer Vision Library, which includes over 500 functions implementing computer vision, image processing and general-purpose numeric algorithms, written in C/C++, and has BSD-like license (that is, absolutely free for academic and commercial use).

3.2 Skin Color based face detection algorithm - Description

This section describes a fast face detection algorithm based on Skin Color detection [12]. This method is both fast and sufficient. There is used a lighting compensation to improve the performance of color-based scheme, and reduce the computation complexity of feature based scheme. Method is effective on facial variations such as dark/bright vision, close eyes, open moth, and a half-profile face.

Figure 10 shows a flow of proposed Skin Color based face detection algorithm.

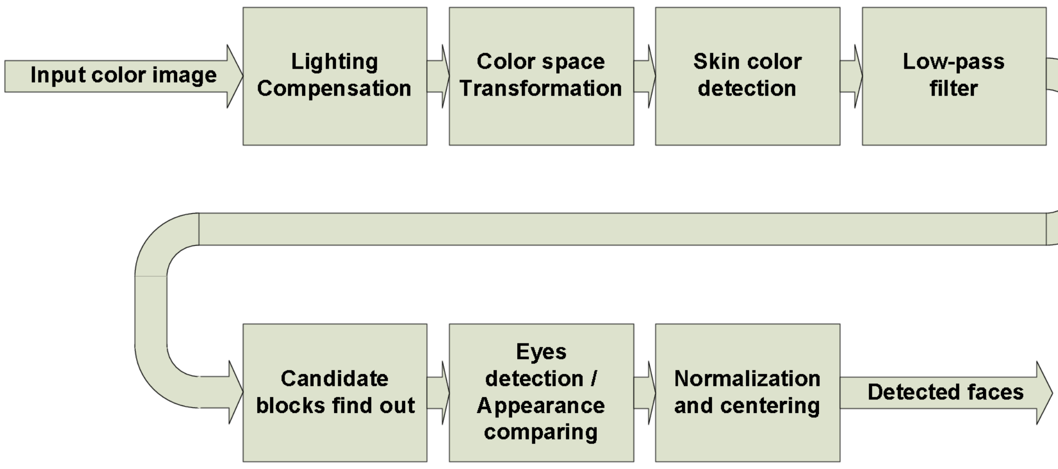

Input color image
Lighting Compensation
Color space Transformation
Skin color detection
Low-pass filter
Candidate blocks find out
Eyes detection / Appearance comparing
Normalization and centering
Detected faces

Figure 10: *The flow of proposed Skin Color based face detection algorithm.*

Description of individual steps:

1.) Color space transformation, Lighting compensation and Skin color detection:

In order to apply to the real-time system, skin-color detection is the first step of face detection. Due to YCbCr color space transform is faster than other approaches, so is used to detect human skin. However, the luminance of every image is different. It results that every image has different color distribution. Therefore, the lighting compensation is based on luminance to modulate the range of skin-color distribution. First step is computing the average luminance Y_{aveg} of input image:

$$Y_{aveg} = \frac{1}{(n \cdot m)} \sum_{i=1;\, j=1}^{n;m} Y_{i,j} \qquad (3)$$

where $Y_{i,j} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$ (Gray-Scale value of RGB color pixel), $Y_{i,j}$ is normalized to the range (0,255), and i, j are the indexes of pixel (size of image is n x m).

According to Y_{aveg}, the compensated image C_{ij} can be determined by following equations:

$$\begin{aligned} R_{ij}' &= (R_{ij})^{T} \\ G_{ij}' &= (G_{ij})^{T} \\ B_{ij} &- no\ change \\ C_{ij} &= \{R_{ij}', G_{ij}', B_{ij}\} \end{aligned} \qquad (4)$$

where:

$$T = \begin{cases} 1.4 & if \quad Y_{aveg} < 64 \\ 0.6 & if \quad Y_{aveg} > 192 \\ 1 & if \quad otherwise \end{cases} \qquad (5)$$

(only R and G are compensated)

Due to chrominance (C_r) can represent human skin well, only Cr is considered as a factor for color space transform to reduce the computation. C_r is defined as follow:

$$C_r = 0.5R' - 0.419G' - 0.081B \qquad (6)$$

In this equation, R' and G' are the most important factors due to their high weight. Thus, only R and G are compensated to reduce computation. According to C_r and experimental experience, the human skin is defined by a binary matrix (array) S_{ij}:

$$S_{ij} = \begin{cases} 0 \; if \quad 10 < C_r < 45 \\ 1 \quad if \quad otherwise \end{cases} \qquad (7)$$

where "0" is the "skin" pixel, and "1" is "non skin" pixel. Figure 11 shows Original color image, and its S_{ij} array (picture of skin detected pixels, where white pixels represent "skin" pixels).

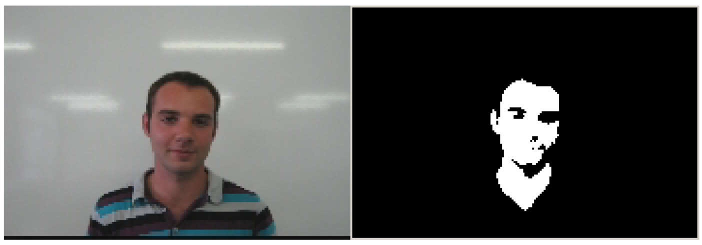


Figure 11: *Original color image, and its skin array S_{ij} (picture of skin detected pixels).*

2.) <u>High frequency noise removing:</u>

In order to remove high frequency noise fast, a low pass filter is implemented by a 5 × 5 mask. First, S_{ij} is segmented into 5×5 blocks, and is calculated how many white points are in a block. Then, every point of a 5 × 5 block is set to white point when the number of white points is greater than half number of total points. On the other hand, if the number of black points is more than a half, this 5 × 5 block is modified to a complete black block. Figure 12 shows an example of high frequency noise removing. Although, this fast filter will bring block effect, it can be disregarded due to that our target is to find where a human skin is. Figure 13 shows noise removing of skin array S_{ij} from Figure 11.

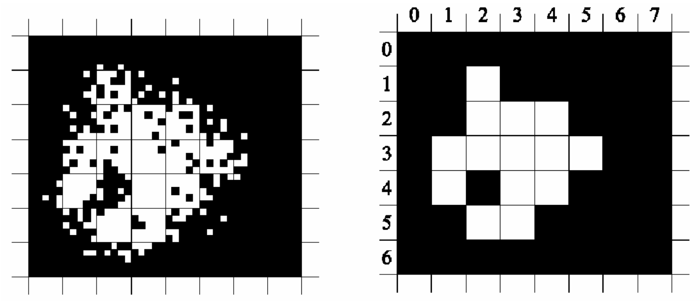


Figure 12: *An example of noise removing by a 5 × 5 mask*

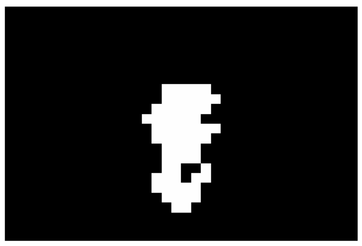


Figure 13: *Noise removing of skin array S_{ij} from Figure 11.*

3.) <u>Candidate blocks find out:</u>

After the step of face localization, several regions are detected. This region may be human face. To correctly detect human face, it is used mask (for example 35 x 35 pixels) which finds areas with the most skin pixels into mask – potential human face. The mask can change its size to find small or large human faces. This algorithm find human faces not at always, because the background can be very complex, so it is better to use another step to find some features of human face to check the potential face to be really human face and not an object with only skin color. Very useful is Eyes detection algorithm or comparing potential face area with typical appearance model of human face.

4.) <u>Eye detection:</u>

Due to the deeper lineaments around human eyes, the existence of human eyes can be detected by the luminance which is slightly darker than average skin-color. The pixels which are around human eyes are defined by E_{hw}:

$$E_{hw} = \begin{cases} 0 & if \; 65 < Y < 80 \\ 1 & othewise \end{cases} \qquad (8)$$

where “0” is pixel around eyes, and “1” is a non eye area.

5.) Normalization and centering:

The last step of Face detection is Normalization, Centering, and other processing of image to get appropriate image of detected face which is good for next use, in this case, face recognition which is sensitive for variance (lighting condition, movement) between detected face and face in database.

Doing normalization means spread values (Gray-Scale) in picture in range [0, 255] for 8-bit Gray-Scale pixels.

Centering means locate face to the center of picture.

3.3 Skin Color based face detection algorithm - Implementation to DSP processor

This section deals with implementation of Skin Color based face detection algorithm described in section 3.2 into DSP processor Blackfin® ADSP-BF561 (section 2.1) with using a development board EZ-KIT Lite® ADSP-BF561 (section 2.2) and development environment VisualDSP++ (section 2.3); all development tools are from Analog Devices, Inc. The language of implementation is C/C++. The Run-time library and DSP library for this processor are also used.

IMPLEMENTATION:

All face detect algorithm is located into one function:

```
void Face_detect_color_based_from_ycbcr_frame(void)
```

Input and output variables are global variables.

Input of this function is array of “unsigned char” values which represent video frame captured from camera:

```
//active field YCbCr: | Cb Y Cr | Y | Cb Y Cr | Y | ....
u8 Frame_ycbcr[NTSC_LINE_WIDTH_ACTIVE*LINES_FROM_FRAME_NTSC];
// 1440 x 525
```

Output of this function is array of "unsigned char" values which represent detected face in Gray-Scale level:

```
u8 Face_one_Y_u8[FACE_DET_SIZE];
// FACE_DET_SIZE = 1225 (35*35) (resizable)
```

Also, output is a new line of 2D array of "double" values which represent new detected face in database of detected faces saved in memory:

```
double Face1_Y[..current-face..][FACE_DET_SIZE];
// MAX_FACES_IN_MEMORY x FACE_DET_SIZE
// FACE_DET_SIZE = 1225 (35*35)
```

<u>Local variables and imported function:</u>

```
f1 = _YCbCrtoRGB; // function: Convert YCbCr to RGB from .asm file:
                  //YCbCrtoRGB.asm

u8 rgb1[3];
u8 ycbcr1[3];
unsigned int px, ln, pxf;
unsigned int i1=0;
unsigned int i2=0;
unsigned int i3=0;
unsigned int i4=0;
unsigned int i5=0;
unsigned int ix5=0;
unsigned int iy5=0;
u8 minY=255;
u8 maxY=0;
u8 maxCn=0;
unsigned int Yavg=0; // average Y
unsigned int count1=0;

typedef struct
{
   unsigned int x;
   unsigned int y;
} Struct_xy;

Struct_xy box_position[MAX_N_BOX]; // position of boxes with poten   //cional
faces - first MAX_N_BOX boxes in the frame

u8 cond1=0;
```

Conversion from 720x480 YCbCr frame to 180x120 RGB frame (:4):

```
//--- Conversion from 720x480 ycbcr frame to 180x120 rgb frame (:4):
    //Frame_ycbcr[1440 x 525] -> rgb_reduced_frame[3x180x120]:

for(ln=(ADI_ITU656_NTSC_ILF1_START-1);ln<(ADI_ITU656_NTSC_ILF1_END);ln=ln+2)
        // lines, every 2nd odd line (4th line in frame)
    {
          for (px=1 ; px<(NTSC_LINE_WIDTH_ACTIVE) ; px=px+8) // columns
          {
                pxf = ln * (NTSC_LINE_WIDTH_ACTIVE) + px; // byte in frame

                if ( ((px+1)%4)==0 ) // every 4th pixel - Y
                {
                      ycbcr1[0]=Frame_ycbcr[pxf]; // Y
                      ycbcr1[1]=Frame_ycbcr[pxf-3]; // Cb
                      ycbcr1[2]=Frame_ycbcr[pxf-1]; // Cr
                }
                else  // every 2th pixel - Y
                {
                      ycbcr1[0]=Frame_ycbcr[pxf]; // Y
                      ycbcr1[1]=Frame_ycbcr[pxf-1]; // Cb
                      ycbcr1[2]=Frame_ycbcr[pxf+1]; // Cr
                }

                f1(ycbcr1, rgb1, 1); // convert YCbCr to RGB

                rgb_reduced_frame[i1+0]=rgb1[0]; // R
                rgb_reduced_frame[i1+1]=rgb1[1]; // G
                rgb_reduced_frame[i1+2]=rgb1[2]; // B

                i1=i1+3;

                //--- Finding min.Y:
                      if (ycbcr1[0]<minY)
                      {
                            minY=ycbcr1[0];
                      }
                //---

                //--- Finding max.Y:
                      if (ycbcr1[0]>maxY)
                      {
                            maxY=ycbcr1[0];
                      }
                //---

                Y_reduced_frame[i2]=ycbcr1[0]; // array of Y
                i2=i2+1;
          }

    }
//---
```

Normalizing of Y frame (array) and Counting average Y:

```
//--- Normalizing of Y frame and Counting average Y
for (i2=0; i2<(X_R_FRAME*Y_R_FRAME); i2=i2+1)
{
    //Y=255.0*(Y-minY)./(maxY-minY);
    Y_reduced_frame[i2]= ( ((Y_reduced_frame[i2]-minY)*255) / (maxY-minY) );
                         // normalizing the Y to be 0..255
    Yavg = Yavg+Y_reduced_frame[i2];
}

Yavg = Yavg / (X_R_FRAME*Y_R_FRAME);
//---
```

Compensation R and G, and computing Cr:

```
float T=1.0;

if (Yavg<64)
{
   T=1.4;
}

if (Yavg>192)
{
    T=0.6;
}

//--- Compensation R and G, and computing Cr (Cr[]=byte_array_reduced[] !!)
    i1=0;
    for (i2=0; i2<(3*X_R_FRAME*Y_R_FRAME); i2=i2+3)
    {
          if (T==1)
          {
                // Cr = 0.5R - 0.419G - 0.081B
                Cr[i1] = ( 127*rgb_reduced_frame[i2+0] -
                           107*rgb_reduced_frame[i2+1] -
                            21*rgb_reduced_frame[i2+2] ) / 255;
                            // computing Cr (Cr = byte_array_reduced[])
          }
          else
          {
                // R'=R^T; G'=G^T; B=B

                rgb_reduced_frame[i2+0]=
                            powf((float)rgb_reduced_frame[i2+0],T);
                      // refill rgb_reduced_frame[] with compensated R' G' B

                rgb_reduced_frame[i2+1]=
                            powf((float)rgb_reduced_frame[i2+1],T);

                rgb_reduced_frame[i2+2]=rgb_reduced_frame[i2+2];

                // Cr = 0.5R' - 0.419G' - 0.081B
                Cr[i1] = ( 0.5*((float)rgb_reduced_frame[i2+0]) -
                        0.419*((float)rgb_reduced_frame[i2+1]) -
```

```
                          0.081*((float)rgb_reduced_frame[i2+2]) );
                                   // counting Cr (Cr = byte_array_reduced[])
            }

            i1=i1+1;
      }
      //---
```

Skin detection:

```
  //--- Skin detection:
      for (i2=0; i2<(X_R_FRAME*Y_R_FRAME); i2=i2+1)
      {
            if ( (Cr[i2]>10) && (Cr[i2]<45) )
            {
                  Skin[i2]=SKIN; // skin pixel;
            }
            else
            {
                  Skin[i2]=NON_SKIN; // non skin pixel
            }
      }
  //---
```

High frequency noise removing - averaging in DIV_N x DIV_N area (5x5):

```
  //--- High frequency noisy removing - averaging in DIV_N x DIV_N area (5x5):
      //Skin[all]=(DIV_N*DIV_N)*Skin_noise_remove[all]

      for (i1=0, iy5=0 ; i1<(Y_R_FRAME); i1=i1+DIV_N, iy5=iy5+1) // in y
      {

            for (i2=0, ix5=0 ; i2<(X_R_FRAME); i2=i2+DIV_N, ix5=ix5+1) // in x
            {
                  count1=0;

                  for (i3=0; i3<(DIV_N); i3=i3+1) // in y (DIV_N area)
                  {
                        for (i4=0; i4<(DIV_N); i4=i4+1) // in x (DIV_N area)
                        {
                              if ( Skin[i1*X_R_FRAME+i2+i3*X_R_FRAME+i4]==SKIN )
                              // if skin pixel
                              {
                                    count1=count1+1;
                              }

                        }
                  }

                  if ( count1 > ( (DIV_N*DIV_N)/2 ) )
                  // if in 5x5 area is more skin pixel, then fill all 5x5
                  // (one area) of skin pixels
                  {
                        Skin_noise_remove[iy5*X_R_REM_FRAME+ix5]=SKIN;
                                                // for testing // Skin area
                        Skin_noise_remove_2D[iy5][ix5]=SKIN; // Skin area
                        }
                        else
                        {
                              Skin_noise_remove[iy5*X_R_REM_FRAME+ix5]=NON_SKIN;
                                                // for testing // non skin area
                              Skin_noise_remove_2D[iy5][ix5]=NON_SKIN;
                                                   // non skin area
                        }

                  }

            }

            //---
```

Finding blocks of potencional faces (Box: 35x35 pixels - 7x7 in noise remove array):

```
      //--- Finding blocks of potencional faces (Box: F_BOX_X x F_BOX_Y):

         maxCn=0;

         for (i1=0; i1<(MAX_N_BOX); i1=i1+1)
         {
               box_position[i1].x = 0;
               box_position[i1].y = 0;
         }

         // Scanning and counting skin pixels in boxes: F_BOX_X x F_BOX_Y:
         // Save position of first MAX_N_BOX (3) with most skin pixels:
         for (iy5=0; iy5<(Y_R_REM_FRAME - F_BOX_Y + 1); iy5=iy5+1)
         {
               for (ix5=0; ix5<(X_R_REM_FRAME - F_BOX_X + 1); ix5=ix5+1)
               {
                     count1=0;
                     for (i3=0; i3<(F_BOX_Y); i3=i3+1)
                     {
                           for (i4=0; i4<(F_BOX_X); i4=i4+1)
                           {
                                 if (Skin_noise_remove_2D[iy5+i3][ix5+i4]==SKIN)
                                 {
                                       count1=count1+1;
                                 }
                           }

                     }

                     if (count1>maxCn) // if new maximum of skin pixels in box
                     {
                           maxCn=count1;

                           for (i1=(MAX_N_BOX-1); i1>0; i1=i1-1)
                           // fill positions of boxes (position left-top of box)
                           {
                                 box_position[i1].x = box_position[i1-1].x;
                                 box_position[i1].y = box_position[i1-1].y;
                           }
```

```
                        box_position[0].x = ix5; // box_position[0]
                                                 // -> position of box with
                                                 // the most skin pixels
                        box_position[0].y = iy5; // box_position[1]
                                                 // -> less pixels, etc., till
                                                 // box_position[MAX_N_BOX]

                    }
                }
            }

            // Mark detected faces to the Skin_noise_remove[] frame:
            ix5=box_position[0].x;
            iy5=box_position[0].y;

            Skin_noise_remove[iy5*X_R_REM_FRAME+ix5]=200;
                    //mark of detected faces (half intesity in grayscale)

            // Create face frame in grayscale from Y_reduced_frame[]:
            for (i1=0; i1<(F_BOX_Y*DIV_N); i1=i1+1)
            {
                    for (i2=0; i2<(F_BOX_X*DIV_N); i2=i2+1)
                    {
                       Face_one_Y_u8[i1*(F_BOX_X*DIV_N)+i2]
                         = Y_reduced_frame[ (iy5*DIV_N+i1)*X_R_FRAME + ix5*DIV_N+i2 ];
                    }
            }

        //---
```

<u>Normalizing of detected faces Face_one_Y_u8[]:</u>

```
      //--- Normalizing of detected faces Face_one_Y_u8[]:

            u8 minYd=255;
            u8 maxYd=0;

            // Finding min.Y and max.Y
            for (i1=0; i1<(FACE_DET_SIZE); i1=i1+1)
            {

                    //--- Finding min.Y:
                    if ( Face_one_Y_u8[i1]<minYd )
                    {
                            minYd = Face_one_Y_u8[i1];
                    }
                    //---

                    //--- Finding max.Y:
                    if ( Face_one_Y_u8[i1] > maxYd )
                    {
                            maxYd = Face_one_Y_u8[i1];
                    }
                    //---
            }
```

```
        // Normalize:
        for (i1=0; i1<(FACE_DET_SIZE); i1=i1+1)
        {
              Face_one_Y_u8[i1]
               = ( ((Face_one_Y_u8[i1]-minYd)*255) / (maxYd-minYd) );
                    // normalizing the Face1_Y[num_faces][i1] to be 0..255

              Face1_Y[num_faces][i1] = (double)Face_one_Y_u8[i1];
        }

    //---
```

3.4 Experimental results

This section contains experimental results of Skin Color based face detection algorithm implemented into DSP processor.

The algorithm is not fully done in this time, is missing Eye detection/Appearance comparing process where is chosen appropriate size of window and centering of face to be in the middle of the window. The face detection works only with face with specific size. Anyway, face detection can works with any size of face, but with static window, so if face would be bigger, in face detection window will not be full face, which is not good for next process – Face recognition.

Processing steps of Face detect algorithm for picture, containing one face, captured from camera:

Figure 14 to Figure 19 show processing steps of Skin Color based face detection algorithm. Images are captured from DSP processor using Windows Application connected to DSP processor via UART or using tool “Image Viewer” in development environment VisualDSP++.

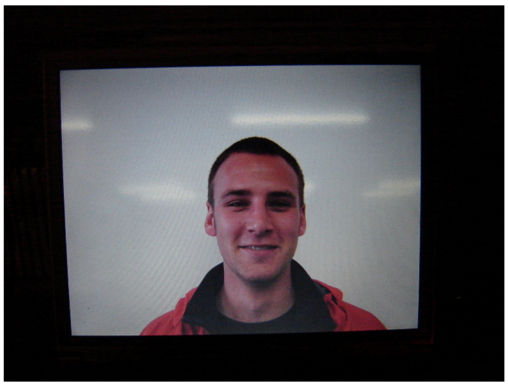


Figure 14: *Original picture captured from camera. (2304x1728 RGB)*

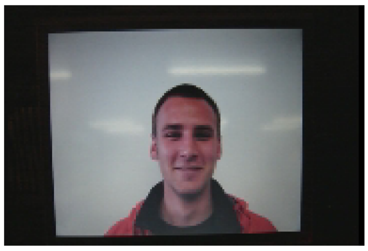


Figure 15: *Image after decimation and lighting compensation. (180x120 RGB)*

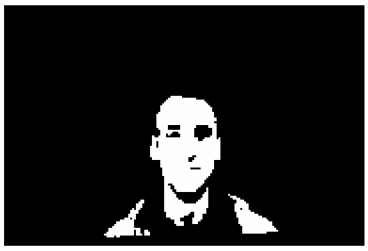


Figure 16: *Skin detection of image. The white pixels are skin pixels. (180x120)*

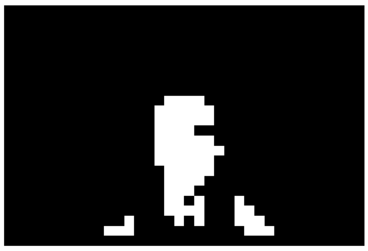


Figure 17: *Image of skin detection after noise removing. (36x24)*

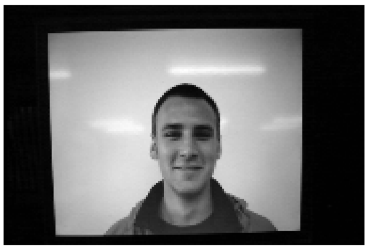


Figure 18: *Image in 8-bit Gray-Scale level after normalization [0, 255]. (180x120)*

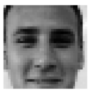


Figure 19: *Detected face in Gray-Scale level after normalization [0, 255]. (35x35)*

<u>Other test:</u>

Figure 20 shows NTSC (ITU-R 656\NTSC) frame captured from camera. The size of frame is 1716x525, but showed image is converted to color view. Image is captured from camera sent to DSP processor and then captured using tool "Image Viewer" in development environment VisualDSP++.

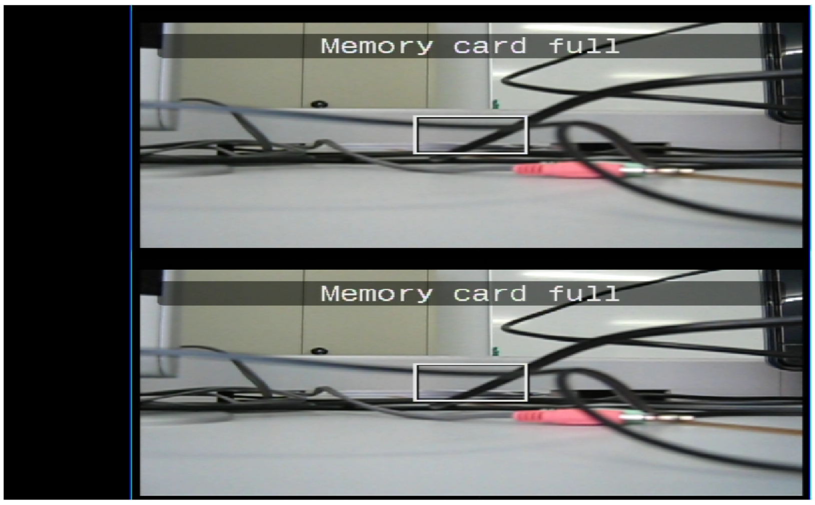


Figure 20: *NTSC frame (ITU-R 656\NTSC) – full, 1716x525, interlaced, captured from camera*

3.5 Summary

This chapter deals with Skin Color based face detection algorithm – description and implementation into DSP processor. In section 3.4 are experimental results of implemented algorithm.

The algorithm works well and can detect face even if some pixels have similar color as human skin (Figure 16 – red jacket). However, algorithm is not fully done, the process of Eye detection/Appearance comparing process is missing. This step assigns choosing appropriate size of window and centering of face to be in the middle of the window. The face detection works only with face with specific size.

Figure 19 shows detected face in Gray-Scale level after normalization from image in Figure 14. Normalization is process where pixels in image are spread and min. value is 0 and max. value is 255 (in Gray-Scale level for 8-bit resolution).

After experimenting with Face recognition based on Eigenfaces (next chapter), is necessary to do another processing steps. The detected face has to be in the middle of window and must have approximately the same size as face in training set. And also, the contrast between face and background should be high. Features of face have to be distinguishable; it can be used, for example, Edge detector, etc.

The Face detection is needful step before Face recognition and quality of detected face impact success of Face recognition.

4 Face recognition based on Eigenfaces (Principal Component Analysis)

This chapter describes Face recognition generally and implementation of Face recognition based on Eigenfaces into DSP processor Blackfin® ADSP-BF561. In the end of this chapter are experimental results of implemented algorithm.

4.1 Introduction to Face recognition

Face recognition [14] is a pattern recognition task performed specifically on faces. It can be described as classifying a face either "known" or "unknown", after comparing it with stored known individuals. It is also desirable to have a system that has the ability of learning to recognize unknown faces.

Computational models of face recognition must address several difficult problems. This difficulty arises from the fact that faces must be represented in a way that best utilizes the available face information to distinguish a particular face from all other faces. Faces pose a particularly difficult problem in this respect because all faces are similar to one another in that they contain the same set of features such as eyes, nose, mouth arranged in roughly the same manner.

In Figure 21 is shown outline of a typical Face recognition system [15].

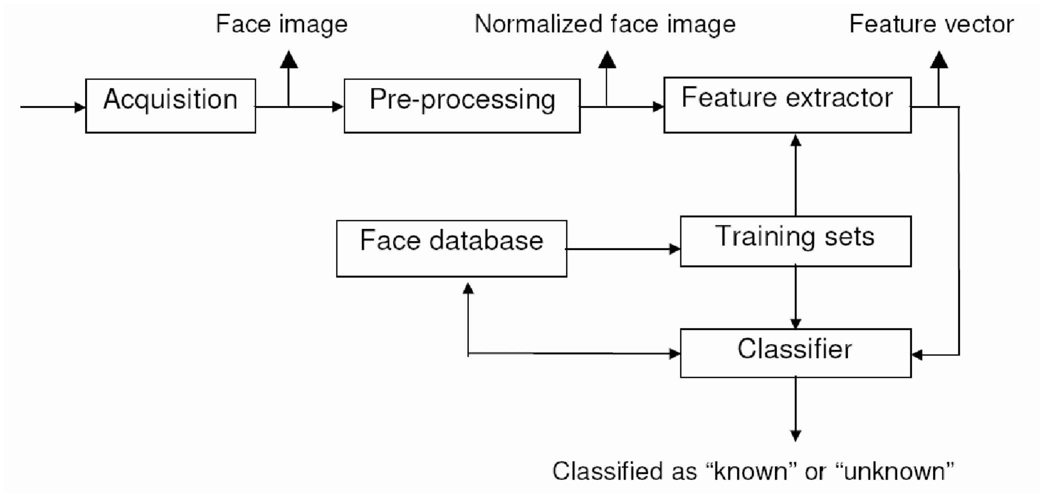


Figure 21: *Outline of a typical face recognition system.*

4.1.1 Approaches to Face Recognition

Main Approaches to Face Recognition are [15]:

§ Feature Based Face Recognition: is based on the extraction of the properties of individual organs located on a face such as eyes, nose and mouth, as well as their relationships with each other. Effective features that can be used in feature based face recognition can be classified as follows:

- First-order features values. Discrete features such as eyes, eyebrows, mouth, chin, and nose, which have been found to be important in face identification and are specified without reference to other facial features, are called first-order features.
- Second-order features values. Another configures set of features, which characterize the spatial relationships between the positions of the first-order features and information about the shape of the face, are called second-order features.

§ Deformable template model. The deformable templates are specified by a set of parameters which uses a priori knowledge about the expected shape of the features to guide the contour deformation process. The templates are flexible enough to change their size and other parameter values, so as to match themselves to the data. The final values of these parameters can be used to describe the features. This method works well regardless of variations in scale, tilt, and rotations.

§ Active contour model (Snake). The active contour or snake is an energy minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges. Snakes lock onto nearby edges, localizing them accurately. Because the snake is an energy minimizing spline, energy functions whose local minima comprise the set of alternative solutions to higher level processes should be designed. Selection of an answer from this set is accomplished by the addition of energy terms that push the model toward the desired solution.

§ Principal component analysis, based on information theory concepts, seek a computational model that best describes a face, by extracting the most relevant information contained in that face without dealing with the individual properties of facial organs such as eyes or mouth.

More information about Face recognition methods and approaches are described in [16].

In this work is used Face recognition method based on Principal component analysis which name is Eigenfaces.

4.2 Principal Component Analysis, Eigenfaces – Description

This section describes Principal Component Analysis and Eigenfaces.

4.2.1 Principal Component Analysis (PCA)

Principal component analysis [16] is standard technique used in statistical pattern recognition and signal processing for data reduction and Feature extraction. As the pattern often contains redundant information, mapping it to a feature vector can get rid of this redundancy and yet preserve most of the intrinsic information content of the pattern. These extracted features have great role in distinguishing input patterns.

A face image in 2-dimension with size $N \times N$ can also be considered as one dimensional vector of dimension N^2. For example, face image with size 35×35 can be considered as a vector of dimension 1225, or equivalently a point in a 1225 dimensional space. An ensemble of images maps to a collection of points in this huge space. Images of faces, being similar in overall configuration, will not be randomly distributed in this huge image space and thus can be described by a relatively low dimensional subspace. The main idea of the principle component is to find the vectors that best account for the distribution of face images within the entire image space. These vectors define the subspace of face images, which be called "face space". Each of these vectors is of length N^2, describes an $N \times N$ image, and is a linear combination of the original face images. Because these vectors are the eigenvectors of the covariance matrix corresponding to the original face images, and because they are face-like in appearance, they can be called "eigenfaces".

4.2.2 Eigenfaces

Eigenfaces approach [15][17] is a principal component analysis method, in which a small set of characteristic pictures are used to describe the variation between face images. Goal is to find out the eigenvectors (eigenfaces) of the covariance matrix of the distribution, spanned by a training set of face images. Later, every face image is represented by a linear combination of these eigenvectors. Evaluation of these eigenvectors is quite difficult for typical image sizes but, an approximation can be made. Recognition is performed by projecting a new image into the subspace spanned by the eigenfaces and then classifying the face by comparing its position in face space with the positions of known individuals.

4.2.3 Calculating Eigenfaces, creating an Eigenface database

In this section is presented scheme for calculating Eigenfaces using Principal Component Analysis (PCA) [18]. A detailed (and more theoretical) description of Eigenfaces can be found in [17].

1.) Training set of faces (face database):

Firstly, there is a training set of faces (face database) F. Training set F is a matrix where each row represents one face (all pixels of face image in Gray-Scale value arranged to 1D array):

$$F = \begin{bmatrix} face_1 \\ \vdots \\ face_P \end{bmatrix} \ , \ face_i = [pixel_1 \ldots pixel_n] \ , \ n = height \times width \qquad (9)$$

Size of F is $p \times n$, where p is a number of faces in training set and n is a number of pixels in face image.

2.) Subtracting the mean face from each faces from training set:

Calculation of mean face of training set:

$$meanF = \frac{1}{p} \cdot \sum_{i=1}^{p} face_i \qquad (10)$$

Data-centering (subtracting the mean face from each faces from training set):

$$\overline{face_i} = face_i - meanF \ , \ i = 1 \ldots p \qquad (11)$$

$$\overline{F} = \begin{bmatrix} \overline{face_1} \\ \vdots \\ \overline{face_P} \end{bmatrix} \quad (12)$$

3.) Calculating the Reduced Covariance matrix:

Reduced Covariance matrix:

$$C' = \overline{F} \cdot \overline{F}^T \quad (13)$$

Size of Reduced Covariance matrix C' is $p \times p$.

The Full Covariance matrix can be calculated:

$$C = \overline{F}^T \cdot \overline{F} \quad (14)$$

Size of Full Covariance matrix C is $n \times n$.

For condition $p << n$ (which is normal case, because number of faces in training set is usually less then their size), is Full Covariance matrix C very large and for later calculation is faster use Reduced Covariance matrix C' (in this case, it is necessary use another step which will be explained later).

The Reduced Covariance matrix C' is real and symmetric (also the Full Covariance matrix C).

4.) Calculating the Eigenvalues and Eigenvectors of Reduced Covariance matrix C':

To calculate the Eigenvalues and Eigenvectors (reduced) of Reduced Covariance matrix C', is necessary to solve this matrix equation (basic eigen-numbers problem):

$$C' \cdot V' = \lambda \cdot V' \quad (15)$$

where V' is matrix of Eigenvectors (reduced) v_i':

$$V' = \begin{bmatrix} v_1' \\ \vdots \\ v_p' \end{bmatrix}, \; V': \; p \times p \quad (16)$$

and λ is diagonal matrix of Eigenvalues λ_i:

$$\lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_p \end{bmatrix} \qquad (17)$$

Eigenvalues and Eigenvectors are calculated from Reduced Covariance matrix, thus Eigenvectors are also reduced (same size as C': $p \times p$).

It is necessary to count Full Eigenvectors V with size $p \times n$:

$$V = V' \cdot \overline{F} \qquad (18)$$

Now Eigenvectors V has size $p \times n$ and represents Eigenvectors of Full Covariance matrix C.

A reason, why is used calculating with Reduced Covariance matrix C' is, that solving the equation $C' \cdot V' = \lambda \cdot V'$ is much faster (even with additional step - counting $V = V' \cdot \overline{F}$) then solving only equation $C \cdot V = \lambda \cdot V$ (with bigger matrixes => more iterations).

Solving the equation $C' \cdot V' = \lambda \cdot V'$ (and also $C \cdot V = \lambda \cdot V$) is quite hard task and is described in section 4.2.5.

Next step is sorting Eigenvectors by values of Eigenvalues from max. to min. (descending order). Eigenvector with the highest value of Eigenvalue the best describes face features of face database and is more significant then Eigenvectors with lower Eigenvalues. It can be used all Eigenvectors or (in case of big number of faces in database) some first Eigenvectors from V.

$$\textit{sorting} \;\; \lambda \;\; -> \;\; \textit{reorganise} \;\; V = \begin{bmatrix} v_1 \\ \vdots \\ v_p \end{bmatrix} \qquad (19)$$

5.) <u>Normalization of Eigenvectors to be a unit vectors:</u>

Normalization v_i:

$$v_i = \frac{v_i}{\|v_i\|} \;, \; i = 1 \ldots p \qquad (20)$$

where:

$$\|v_i\| = \sqrt{\sum_{i=1}^{p} v_i^{\,2}} \ , \ (\textit{ for real } v_i \,) \tag{21}$$

Now, matrix V (Eigenvectors) represents base of Eigenspace.

6.) <u>Transformation of training set $\overline{F}$ onto the Eigenspace:</u>

All centered faces $\overline{face_i}$ from training set $\overline{F}$ are projected onto the Eigenspace by the calculated Eigenvectors V :

$$\tilde{f}_i = \overline{face_i} \cdot V^T \ , \ i = 1 \ldots p \tag{22}$$

$$\tilde{F} = \begin{bmatrix} \tilde{f}_1 \\ \vdots \\ \tilde{f}_p \end{bmatrix} \tag{23}$$

$\tilde{F}$ represents so-called Eigenfaces database (training set projected onto the Eigenspace). Eigenfaces database contains extracted important face features of faces from training set.

4.2.4 Recognition process of unknown face

Unknown face must have the same size as faces from training set:

$$unknown\_face = [pixel_1 \ldots pixel_n] \ , \ n = height \times width \tag{24}$$

Data-centering (subtracting the mean face of training set calculated before):

$$\overline{u} = unknown\_face - meanF \tag{25}$$

Unknown face is projected onto the Eigenspace by Eigenvectors V calculated before:

$$\tilde{u} = \overline{u} \cdot V^T \tag{26}$$

Recognition process is based on comparing of Unknown face projected onto the Eigenspace by the calculated Eigenvectors with all faces from training set projected onto the same Eigenspace (Eigenfaces database) by the same Eigenvectors.

For classifying which face from training set is the most similar with unknown face, it can be used metric distance between unknown sub-space (unknown face in Eigenspace) and Eigenfaces database, for example, Euclidian distance:

$$d_E\left(\tilde{u}, \tilde{f}_i\right) = \sqrt{\sum_{j=0}^{k-1}\left(\tilde{u}[j] - \tilde{f}_i[j]\right)^2} \quad , \quad i = 1 \ldots p \qquad (27)$$

$k = p$ or $k < p$ *(depends of how many Eigenvectors were chosen)*

Also, Threshold can be chosen for classifying of unknown face. If $d_E\left(\tilde{u}, \tilde{f}_i\right) < Threshold$, then $face_i$ from training set is probably unknown face. Choosing Threshold is very important step, and is determined experimentally depending on training set.

4.2.5 Jacobi's cyclic method: Calculation Eigenvalues and Eigenvectors

For calculation Eigenvalues and Eigenvectors of Covariance matrix is used Jacobi's cyclic method of which source code written in C is placed at [20]. Another information and source code of some numerical algorithms can be found at [19].

The Covariance matrix used in this project (section 4.2) is always real and symmetric matrix. For a real symmetric matrix all the eigenvalues are real. If all the eigenvalues and corresponding eigenvectors are wanted then Jacobi's cyclic method can be used.

Jacobi's cyclic method:

The Jacobi's cyclic method [19] is written into a one function (which can be found in source codes of this project):

```
void Jacobi_Cyclic_Method( double eigenvalues[ ], double
*eigenvectors, double *A, int n )
```

Given the n × n real symmetric matrix A, the routine Jacobi_Cyclic_Method calculates the eigenvalues and eigenvectors of A by successively sweeping through the matrix A annihilating off-diagonal non-zero elements by a rotation of the row and column in which the non-zero element occurs. The input matrix A is modified during the process. The eigenvalues are returned in the array eigenvalues which should be dimensioned at least n in the calling routine. The eigenvectors are returned in the n×n matrix eigenvectors, the i_{th} column being the eigenvector corresponding the the i_{th} eigenvalue, eigenvalue[i].

4.3 Face recognition based on Eigenfaces - Implementation to DSP processor

This section deals with implementation of Face recognition based on Eigenfaces described in section 4.2 into DSP processor Blackfin® ADSP-BF561 (section 2.1) with using a development board EZ-KIT Lite® ADSP-BF561 (section 2.2) and development environment VisualDSP++ (section 2.3); all development tools are from Analog Devices, Inc. The language of implementation is C/C++. The Run-time library and DSP library for this processor are also used.

Figure in Appendix B1 shows Project diagram, where are demonstrated the development board and implemented algorithms including Face recognition algorithm.

<u>IMPLEMENTATION:</u>

All Face recognition algorithms are located to the functions:

```
void Create_eigenface_database(void)
```

- for Calculating Eigenfaces, creating an Eigenface database.

```
void Face_recognition(void)
```

- for Recognition process of unknown face.

Input and output variables are global variables.

4.3.1 Calculating Eigenfaces, creating an Eigenface database

```
void Create_eigenface_database(void)
```

Input of this function is 2D array - training set of faces (of “double” values), where each row represents one face:

```
double Face1_Y[MAX_FACES_IN_MEMORY][FACE_DET_SIZE];
// FACE_DET_SIZE = 1225 (35*35)
```

Output of this function is 2D array - Eigenfaces database (training set projected onto the Eigenspace by the calculated Eigenvectors), of “double” values, where each row represents one face projected onto the Eigenspace:

```
double
Eigeface_database[MAX_FACES_IN_MEMORY][MAX_FACES_IN_MEMORY];
// Size of array is: n_of_faces x n_of_faces
```

Local variables:

```
unsigned int pix1=0;
unsigned int f1=0;
unsigned int a1=0;
unsigned int pom1=0;
double acc_double=0;
unsigned int pom_p[NUMBER_FACES];
```

Mean face from training set of faces (detected faces):

```
//--- Mean face from detected faces:
    for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
    {
           Mean_face[pix1]=0;

           for (f1=0; f1<NUMBER_FACES; f1=f1+1)
           {
                  Mean_face[pix1] = Face1_Y[f1][pix1] + Mean_face[pix1];
           }

           Mean_face[pix1] = ( Mean_face[pix1] / NUMBER_FACES );
    }
//---
```

Subtract the mean face from every face from training set:

```
//--- Subtract the mean face:
    for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
    {
           for (f1=0; f1<NUMBER_FACES; f1=f1+1)
           {
                  Face1_Y[f1][pix1] = Face1_Y[f1][pix1] - Mean_face[pix1];
           }
    }
//---
```

Create transposed matrix of Face1_Y[][]:

```
//--- Create transposed matrix of Face1_Y[][]:
    for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
    {
          for (f1=0; f1<NUMBER_FACES; f1=f1+1)
          {
                Face1_Y_transpose[pix1][f1] = Face1_Y[f1][pix1];
          }
    }
//---
```

Create Reduced Covariance matrix C':

```
//--- Create Covariance matrix C' (modified):
    matmmlt(*Face1_Y, NUMBER_FACES, FACE_DET_SIZE, *Face1_Y_transpose,
            NUMBER_FACES, *Covariance_matrix_m);
    // matrix * matrix multiplication: Covariance_matrix_m[][]
    // = Face1_Y[][] * Face1_Y_transpose[][] ( NUMBER_FACES x NUMBER_FACES )
//---
```

Calculation Eigenvectors (Reduced, NUMBER_FACES x NUMBER_FACES) and Eigenvalues (NUMBER_FACES x 1) from Reduced Covariance matrix C' (Jacobi's cyclic method):

```
//--- Counting Eigenvectors V' and Eigenvalues λ from Covariance matrix C'
(Jacobi's cyclic method):

// A*V = l*V
// Jacobi_Cyclic_Method(eigenvalues, (double*)eigenvectors, (double*)A, N);
// ( Covariance_matrix_m[][] * Eigenvectors_m[][]
//   = Eigenvalues[] * Eigenvectors_m[][] )

    Jacobi_Cyclic_Method( Eigenvalues, (double*)Eigenvectors_m,
                          (double*)Covariance_matrix_m, NUMBER_FACES );
          // Jacobi's cyclic method in "jacobi_cyclic_method.c"

//---
```

Creating Full Eigenvectors (NUMBER_FACES x FACE_DET_SIZE):

```
//--- Create true eigenvectors (NUMBER_FACES x FACE_DET_SIZE):
    matmmlt(*Eigenvectors_m, NUMBER_FACES, NUMBER_FACES, *Face1_Y,
            FACE_DET_SIZE, *Eigenvectors);
          // matrix * matrix multiplication:
          // Eigenvectors[][] = Eigenvectors_m[][] * Face1_Y[][]
//---
```

Sorting of Eigenvalue (max -> min) and saving original indexes:

```
//--- Sorting of eigenvalue (max -> min) and save original indexes:
    for (f1=0; f1<NUMBER_FACES; f1=f1+1)
    {
          P_Eigenvalues[f1] = Eigenvalues[f1];
                // Saving unsorted array of Eigenvalues[]
    }

    qsort (Eigenvalues, NUMBER_FACES, sizeof(Eigenvalues[0]), compare_double);
          // sort Eigenvalues[] in (min -> max) order
    for (f1=0; f1<NUMBER_FACES; f1=f1+1)
    {
          for (a1=0; a1<NUMBER_FACES; a1=a1+1)
          {
                if ( Eigenvalues[f1] == P_Eigenvalues[a1] )
                {
                      pom_p[NUMBER_FACES-1-f1]=a1;
                          // index of unsorted Eigenvalues in
                          // sorted Eigenvalues
                }

          }
    }
//---
```

Normalization of Eigenvectors to be a unit vectors:

```
//--- Normalization of Eigenvectors to be a unit vectors:
    for (f1=0; f1<NUMBER_FACES; f1=f1+1)
    {
          acc_double=0;

          for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
          {
                acc_double = acc_double +
                 + ( Eigenvectors[f1][pix1] * Eigenvectors[f1][pix1] );
          }

          acc_double = sqrt(acc_double);

          for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
          {
                Eigenvectors[f1][pix1] = (Eigenvectors[f1][pix1]/acc_double);
          }
    }
//---
```

Creating transposed matrix of Eigenvectors[][] and sort by sorted values of eigenvalues[]:

```
//---
    for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
    {
          for (f1=0; f1<NUMBER_FACES; f1=f1+1)
          {
                pom1 = pom_p[f1];
                      // conversion index for sorting Eigenvectors[][]
                      // by sorted values of eigenvalues[]
                Eigenvectors_transpose[pix1][f1] = Eigenvectors[pom1][pix1];
                      // transponsing
          }
    }
//---
```

Transformation of training set onto the Eigenspace:

```
//--- Creating Eigenface database:
    for (f1=0; f1<NUMBER_FACES; f1=f1+1)
    {
          for (a1=0; a1<FACE_DET_SIZE; a1=a1+1)
          {
                One_face[0][a1] = Face1_Y[f1][a1];
          }

          matmmlt(*One_face, 1, FACE_DET_SIZE, *Eigenvectors_transpose,
                  NUMBER_FACES, *One_eigenface);
                      // matrix * matrix multiplication:
                      // One_eigenface[]
                      // = Face1_Y[1][] * Eigenvectors_transpose[][]

          for (a1=0; a1<NUMBER_FACES; a1=a1+1)
          {
                Eigenface_database[f1][a1] = One_eigenface[0][a1];
          }
    }
//---
```

4.3.2 Recognition process of unknown face

```
void Face_recognition(void)
```

Input of this function is a new detected face in database of detected faces (of “double” values):

```
double Face1_Y[..current-face..][FACE_DET_SIZE];
// FACE_DET_SIZE = 1225 (35*35)
```

Output of this function is array of “double” values, which represent Distances (Euclidian) between Unknown face projected onto the Eigenspace and all faces from training set projected onto the Eigenspace by the calculated Eigenvectors:

```
double Distance_recognition[MAX_FACES_IN_MEMORY];
// number of Distences are number of faces in training set
```

Local variables:

```
unsigned int pix1=0;
unsigned int f1=0;
unsigned int a1=0;
unsigned int pom1=0;
double acc_double=0;
```

Subtract the mean face from every face from training set:

```
//--- Subtract the mean face (current_face - Mean_face[]):
    for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
    {
           One_face[0][pix1] = Face1_Y[num_faces][pix1] - Mean_face[pix1];
    }
//---
```

Transformation of unknown face onto the Eigenspace (by the same Eigenvectors as in creating an Eigenfaces database :

```
//---
    matmmlt(*One_face, 1, FACE_DET_SIZE, *Eigenvectors_transpose,
            NUMBER_FACES, *One_eigenface);
                // matrix * matrix multiplication: One_eigenface[]
                // = Face1_Y[1][] * Eigenvectors_transpose[][]

//---
```

Comparing the unknown face in Eigenspace with Eigenfaces database by Euclidean Distances between them:

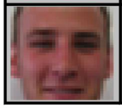
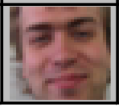
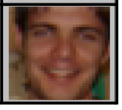
```
//--- Classifying the unknown faces - Euclidean Distance (distance between
// unknown eigenface and all eigenfaces from Eigenface database):
    for (f1=0; f1<NUMBER_FACES; f1=f1+1)
    {
           acc_double=0;

           for (a1=0; a1<NUMBER_FACES; a1=a1+1)
           {
                 acc_double = acc_double +
                  + ( (One_eigenface[0][a1] - Eigenface_database[f1][a1])
                  * (One_eigenface[0][a1]-Eigenface_database[f1][a1]) );
           }

           Distance_recognition[f1] = sqrt(acc_double);
                 // Euclidean Distance between unknown eigenface
                 // and f1-th eigenface from Eigenface database

           Distance_recognition_u8[f1] = (u8) ( Distance_recognition[f1] / 4 );
                 // convert to be seen in "u8" range
    }

    for (f1=NUMBER_FACES; f1<MAX_FACES_IN_MEMORY; f1=f1+1)
    {
           Distance_recognition[f1]=DBL_MAX;
                 // fill rest of array with max-distance number
           Distance_recognition_u8[f1]=255;
    }
//---
```

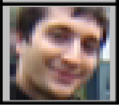
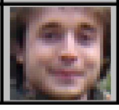
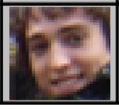
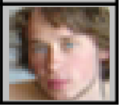
4.4 Experimental results

This section contains experimental results of Face recognition based on Eigenfaces implemented into DSP processor.

Table 1 shows faces from Training set (face database) of 15 faces. From these faces was created Eigenface database - Training set projected onto the Eigenspace by the calculated Eigenvectors.

Table 1: *Faces from Training set:*

01	02	03	04	05	06	07	08
							

09	10	11	12	13	14	15
						

Recognition process is based on comparing of Unknown face projected onto the Eigenspace by the calculated Eigenvectors with all faces from Training set projected onto the Eigenspace by the same Eigenvectors.

For testing purpose, were created modified faces of Training set. Modification was different:

- § covered mouth (results in Appendix A1)
- § added moustache (results in Appendix A2)
- § shifting of 1 pixel to left-down (results in Appendix A3)
- § rotation of 3 degree in the direction of clock (results in Appendix A4)
- § decrease size of faces (95% in x and 95% in y) (results in Appendix A5)

§ testing one face 02 with some modifies (results in Appendix A6_1 and Appendix A6_2)

§ testing one face 05 with some modifies (results in Appendix A7)

For classifying of similarity were chosen Euclidian distances between Unknown face projected onto the Eigenspace and all faces from Training set projected onto the Eigenspace.

The less is distance then the Unknown face is more similar to face from Training set.

Tables in Appendix A1 to Appendix A7 show Euclidian distances in these tests. For good similarity (same faces) is distance around 10.0 – 50.0, for bad similarity (different faces) is distance > 200.0 . For different faces with no similarity is distance > 1000.0.

In these tables, the red highlighted means incorrect recognition (smaller distance for this face then distance of same or modified face). If there are one or more red highlighted distances in line, then the recognition is incorrect.

Tests, with changes, such as shifting, rotation, and decrease size of faces to be recognized, are shown in Appendix A3 up to Appendix A5 for no big change. If faces will be more changed as original faces from training set, the results of Face recognition will be not satisfied (high number of red highlighted distances – incorrect recognition), because Face recognition based on Eigenfaces is very sensitive to this type of changes.

4.5 Summary

This chapter deals with Face recognition based on Eigenfaces – description and implementation into DSP processor. In section 4.4 are experimental results of implemented algorithm.

Face recognition based on Eigenfaces uses principles of Principal Component Analysis. The main idea of the principle component is to find the vectors that best account for the distribution of face images within the entire image space. These vectors define the subspace of face images, which be called “face space” or “Eigenspace”. Recognition process is based on comparing of Unknown face projected onto the Eigenspace by the calculated Eigenvectors with all faces from training set projected onto the Eigenspace by the same Eigenvectors.

Experiment of Face recognition was done with training set of faces showed in Table 1. From this training set was calculated Eigenfaces database. As Unknown face were used modified faces from training set. Modifications were chosen: covered mouth, added moustache, shifting of 1 pixel to left-down, rotation of 3 degree, decrease size, and some modifies with one face 02 and face 05. Results of this experiment are shown in Appendix A1 up to Appendix A7. There are Euclidian distances which represent metric distances between Unknown face projected onto the Eigenspace and all faces from Training set projected onto the Eigenspace.

From these results follows characteristics and requirements of Face recognition based on Eigenfaces. Pre-step of Face recognition is Face detection. Quality of detected face impacts success of Face recognition.

Face recognition based on Eigenfaces is very sensitive with these factors:

§ movement, resizing and rotation of face in face image compared with faces in training set

§ Change contrast of image (face vs. background or between face features)

§ Change of Gray-Scale level of some pixels (for example, results are different for white glasses and black glasses, even, the glasses are same and cover the same part of face)

Follow of these, Face detection, as a pre-step of Face recognition, is very important and Face detection should arrange detected face with consideration of these factors.

These factors are same for preparing faces of training set. Besides, creating training set (face database) is also very important process. It can be used more views of same face to reduce wrong recognitions. Furthermore, number of faces in training set and size of face image must be consider to have fast and reliable face recognition system.

5 Other implementations and Windows Application

This chapter describes other project code for DSP processor Blackfin® ADSP-BF561 (section 2.1) with using a development board EZ-KIT Lite® ADSP-BF561 (section 2.2) and development environment VisualDSP++ (section 2.3); all development tools are from Analog Devices, Inc.

Also, there is described designed Windows Application, which communicates with DSP processor via UART. The Windows Application helped with designing and implementing algorithms and now, is used as supportive application for loading faces into DSP processor, showing result of Face detection and Face recognition, etc.

5.1 Other algorithms for DSP processor

This section describes algorithms (code) such as Video frame capturing from camera, Displaying frame on monitor, Communication with PC via UART, Converting from YCbCr mode to RGB mode, and others. These codes make functionality with environment to implemented algorithms before (in chapter 3 and 4).

5.1.1 Video frame capturing from camera

Video format for video streaming is ITU-R 656\NTSC (information about this video format is describes in section 1.3).

Program in DSP processor is using only active NTSC frame. The size of frame is 1440x480 (resolution: 720x480) divided into 2 fields, first field with even lines and second field with odd lines (interlaced system). There is used only first field, every second line and every fourth pixel in line (for face detection is used decimated frame 180x120).

The source code of Video frame capturing from camera:

```
void CaptureFrame(
    ADI_DCB_HANDLE  DCBManagerHandle,
    ADI_DMA_MANAGER_HANDLE  DMAManagerHandle,
    ADI_DEV_MANAGER_HANDLE  DeviceManagerHandle,
    ADI_DEV_2D_BUFFER  *pBuffer2D
){
```

```
// table of configuration values for the PPI on input
ADI_DEV_CMD_VALUE_PAIR InboundConfigurationTable [] = {
      { ADI_DEV_CMD_SET_DATAFLOW_METHOD,  (void *)ADI_DEV_MODE_CHAINED  },
      { ADI_PPI_CMD_SET_CONTROL_REG, (void *)0x00c0 },
                  // Active field only, Field 1 & 2, YcbCr - all
      { ADI_PPI_CMD_SET_LINES_PER_FRAME_REG,(void*)LINES_FROM_FRAME_NTSC},
                  // ADI_ITU656_NTSC_HEIGHT
      { ADI_DEV_CMD_END, NULL },
};

ADI_DEV_DEVICE_HANDLE   DeviceHandle;     // handle to the device driver

// turn on first LED
ezTurnOnLED(0);

// enable the video decoder (7183)
ezEnableVideoDecoder();

// give the decoder time to sync
ezDelay(300);

// since we're doing input, have the PPI driver generate a callback
// when the buffer
// has completed processing.
// The pArg parameter to the callback function will be
// whatever we set the CallbackParameter value to.
pBuffer2D->CallbackParameter = pBuffer2D;

// open the PPI driver for input
ezErrorCheck(adi_dev_Open(DeviceManagerHandle, &ADIPPIEntryPoint,
             DECODER_PPI, NULL, &DeviceHandle, ADI_DEV_DIRECTION_INBOUND,
             DMAManagerHandle, DCBManagerHandle, Callback));

// configure the PPI driver with the values
// from the inbound configuration table
 ezErrorCheck(adi_dev_Control(DeviceHandle, ADI_DEV_CMD_TABLE,
             InboundConfigurationTable));

// give the PPI driver the buffer to process
ezErrorCheck(adi_dev_Read(DeviceHandle, ADI_DEV_2D,
             (ADI_DEV_BUFFER *)pBuffer2D));

// tell the PPI driver to enable data flow
ezErrorCheck(adi_dev_Control(DeviceHandle, ADI_DEV_CMD_SET_DATAFLOW,
                             (void *)TRUE));

// wait till the buffer has been processed
while (pBuffer2D->ProcessedFlag == FALSE) ;

// delay
ezDelay(300);

// close the PPI driver
ezErrorCheck(adi_dev_Close(DeviceHandle));

// turn off first LED
```

```
    ezTurnOffLED(0);

    // return
}
```

Program uses buffer type of structure which contains pointer to array, size of array, Callback Parameter, or pointer to next buffer when this buffer is processed:

```
// Populate the buffer that we'll use for the PPI input and output
//Frame_ycbcr[1440 * ADI_ITU656_NTSC_HEIGHT]; // 1440 x 525
Buffer2D.Data = Frame_ycbcr;
Buffer2D.ElementWidth = 2;
Buffer2D.XCount = (NTSC_LINE_WIDTH_ACTIVE / 2);
Buffer2D.XModify = 2;
Buffer2D.YCount = LINES_FROM_FRAME_NTSC;
Buffer2D.YModify = 2;
Buffer2D.CallbackParameter = NULL;
Buffer2D.pNext = NULL;
```

After, when is buffer processed, can be called Callback function, which can be used for more buffer-processed action (UART, etc.):

```
static void Callback(
    void *AppHandle,
    u32  Event,
    void *pArg)
{

    static unsigned int Counter = 0;
          // count the number of input buffers processed
    ADI_DEV_BUFFER *pBuffer;
          // pointer to the buffer that was processed

    // CASEOF (event type)
    switch (Event) {

          // CASE (buffer processed)
          case ADI_DEV_EVENT_BUFFER_PROCESSED:

                // point to the buffer
                pBuffer = (ADI_DEV_BUFFER *)pArg;

                // increment our counter
                Counter++;

                if (pBuffer == (ADI_DEV_BUFFER *)&OutboundBuffer)
                      // if processed OutboundBuffer - UART
                {
                      uart_processed1=1; // UART was processed
                }

                break;

          // CASE (an error)
          case ADI_DEV_EVENT_DMA_ERROR_INTERRUPT:
          case ADI_PPI_EVENT_ERROR_INTERRUPT:

                // turn on all LEDs and wait for help
                ezTurnOnAllLEDs();
                while (1) ;

    // ENDCASE
    }

    // return
}
```

Development board EZ-KIT Lite ADSP-BF561 contains video decoder and video encoder, which provides channels of analog video input and output (more information in section 2.2).

5.1.2 Communication with PC via UART

Setting up the serial communication via UART is similar as Video frame capturing or Displaying frame at monitor. Instead of video decoder device is used UART device and buffer point to array to be processed by UART.

The source code of Communication with PC via UART:

```
  void Init_and_Send_buffer_to_UART(
      ADI_DCB_HANDLE                DCBManagerHandle_u,
      ADI_DMA_MANAGER_HANDLE  DMAManagerHandle_u,
      ADI_DEV_MANAGER_HANDLE  DeviceManagerHandle_u,
      ADI_DEV_1D_BUFFER             *pBuffer1D
  )
  {
      // table of configuration values for the UART
      ADI_DEV_CMD_VALUE_PAIR ConfigurationTable [] = {
            // configuration table for the UART driver
            { ADI_DEV_CMD_SET_DATAFLOW_METHOD, (void *)ADI_DEV_MODE_CHAINED },
            { ADI_UART_CMD_SET_DATA_BITS, (void *)8 },
            { ADI_UART_CMD_ENABLE_PARITY, (void *)FALSE },
            { ADI_UART_CMD_SET_STOP_BITS, (void *)1 },
            { ADI_UART_CMD_SET_BAUD_RATE, (void *)BAUD_RATE },
            { ADI_UART_CMD_SET_LINE_STATUS_EVENTS, (void *)TRUE },
            { ADI_DEV_CMD_END, NULL },
      };

      // Handle to the UART driver
      ADI_DEV_DEVICE_HANDLE   DriverHandle_u;

      // turn on 3. LED
      ezTurnOnLED(2);

      // callback after buffer processed enabled
       pBuffer1D->CallbackParameter = pBuffer1D; //NULL; //pBuffer1D;
```

```
    //
    uart_processed1=0;

    // open the UART driver for bidirectional data flow
    ezErrorCheck(adi_dev_Open(DeviceManagerHandle_u, &ADIUARTEntryPoint, 0,
                 NULL, &DriverHandle_u, ADI_DEV_DIRECTION_BIDIRECTIONAL, NULL,
                 DCBManagerHandle_u, Callback));

    // configure the UART driver with the values from the configuration table
    ezErrorCheck(adi_dev_Control(DriverHandle_u, ADI_DEV_CMD_TABLE,
                 ConfigurationTable));


    // send the outbound buffer 1D
    ezErrorCheck(adi_dev_Write(DriverHandle_u, ADI_DEV_1D,
                 (ADI_DEV_BUFFER *)pBuffer1D));

    // enable data flow
    ezErrorCheck(adi_dev_Control(DriverHandle_u, ADI_DEV_CMD_SET_DATAFLOW,
                                 (void *)TRUE));

    // wait till the buffer has been processed
    while (pBuffer1D->ProcessedFlag == FALSE) ;

    uart_processed1=0;

    // wait for sending last byte before close UART driver
    ezDelay(300);

    // close the UART driver
    ezErrorCheck(adi_dev_Close(DriverHandle_u));


    ezDelay(300); // delay for LED

    // turn off 3. LED
    ezTurnOffLED(2);

} // End of function: Init_and_Send_buffer_to_UART
```

<u>The buffer for sending via UART:</u>

```
// Buffer for send array via UART
OutboundBuffer3.Data = Array_u8;
OutboundBuffer3.ElementCount = (FACE_DET_SIZE); //(NUMBER_FACES);
OutboundBuffer3.ElementWidth = 1;
OutboundBuffer3.CallbackParameter = &OutboundBuffer3;
OutboundBuffer3.ProcessedFlag = FALSE;
OutboundBuffer3.pNext = NULL;
```

Receiving buffer from UART is very similar.

5.1.3 Converting from YCbCr mode to RGB mode

Color mode of pixels in video format is YCbCr. For Skin Color based face detection algorithm is needed use RGB color mode.

Converting from YCbCr mode to RGB mode is executed by extern function written in assembler in .asm file: "YCbCrtoRGB.asm" which is imported to the project. File is written by Analog Devices, Inc. and can be seen in project source code. Calculating this conversion is not hard, but using optimized asm code is faster.

5.2 Windows Application

A Windows Application, which was designed, helped with designing and implementing algorithms and now, is used as supportive application for loading faces into DSP processor, showing result of Face detection and Face recognition, etc.

The Windows Application communicates with PC via UART. It is written in Microsoft Visual Studio 2008 (student license). The language of source code is Visual C++.

The Windows Application can read color image from file and convert to the Gray-Scale level and normalize it. Then can send picture to DSP processor via UART. Also, can shows detected face from DSP processor and shows result of Face Recognition by writing Distance numbers and showing recognized face from training set.

The best way, how to test Face Recognition, is create pictures of one face with 35x35 size and send this training set (face database) to DSP processor and show Distance numbers after Face Recognition process.

6 Conclusion

This work deals with development of algorithms for image processing, such as Face detection and Face recognition algorithms, and with implementation of these algorithms into DSP processor Blackfin® ADSP-BF561 with using a development board EZ-KIT Lite® ADSP-BF561 and development environment VisualDSP++; all development tools are from Analog Devices, Inc.

Description, details about implementation, experimental results and summary of Skin Color based face detection algorithm and Face recognition based on Eigenfaces are shown in Chapter 3 (Face detection) and Chapter 4 (Face recognition). Experimental results of Face detection are shown in section 3.4. Experimental results of Face recognition are shown in Appendix A1 to Appendix A7 and in section 4.4.

Algorithms work well with appropriate results; however, Face detection is not fully done, the process of Eye detection/Appearance comparing process is missing. This step assigns choosing appropriate size of window and centering of face to be in the middle of the window. The face detection works only with faces with specific size.

The algorithms are working in DSP processor, and also can work under the Windows, because they are written in C/C++; some function (matrix multiplication, sorting, and others) are part of library for DSP processor, so this functions need to implement from library for Windows.

Also, I have developed a Windows Application, which is used as supportive application for loading faces into DSP processor, and showing result of Face detection and Face recognition. The Windows Application communicates with DSP processor via UART. The Windows Application is developed in Microsoft Visual Studio 2008, in Visual C++.

Figure in Appendix B1 shows Project diagram, where are demonstrated the development board and implemented algorithms including Face recognition based on Eigenfaces algorithm. Figure 10 shows implemented Skin Color based face detection algorithm.

References and bibliography

[1] Gonzalez, R. C., Woods, R. E., "Digital Image Processing", Prentice Hall, New Jersey 2002, Second edition

[2] Acharya, T., Ray, A. K., "Image processing : principles and applications", John Wiley & Sons, New Jersey 2005

[3] Dobeš, M., "Zpracování obrazu a algoritmy v C#", BEN - technická literatura, Praha 2008, 1. vydání

[4] "ADSP-BF561 Blackfin® Processor Hardware Reference", Analog Devices, Inc., Reference document, [cit. 2/6/2009], Available from WWW: http://www.analog.com/processors

[5] Keith, J., "YCbCr to RGB Considerations", AN9717, Intersil Americas Inc., Application Note

[6] "ADSP-BF561", Analog Devices, Inc., Datasheet, [cit. 2/6/2009], Available from WWW: http://www.analog.com/processors

[7] "ADSP-BF561 EZ-KIT Lite® Evaluation System Manual", Datasheet, [cit. 2/6/2009], Available from WWW: http://www.analog.com/processors

[8] Yang,M. H., Kriegman, D. J., Ahuja, N., "Detecting Faces in Images: A Survey", IEEE Transactions on pattern analysis and machine intelligence, vol. 24, no. 1, pp. 34-58, January 2002

[9] Viola, P., Jones, M. J., Fast Multi-view Face Detection, Mitsubishi Electric Research Laboratories, TR2003-096, August 2003.

[10] OpenCV library download: http://sourceforge.net/projects/opencvlibrary

[11] OpenCV library – information: http://opencv.willowgarage.com

[12] Pai, Y. T., Ruan, S. J., Shie, M. Ch., Liu, Y. Ch., "A simple and accurate color face detection algorithm in complex background", IEEE ICME, pp. 1545-1548, 2006

[13] "VisualDSP++ 5.0 User's Guide", User's Guide, [cit. 2/6/2009], Available from WWW: http://www.analog.com/processors

[14] Atalay, I., "Face recognition using eigenfaces", MSc. thesis, Istanbul Technical University, 1996

[15] Atalay, I., "Brief introduction to : Pattern Recognition, Face Recognition, Face Recognition Using EigenFaces", article

[16] Delac, K., Grgic, M., "Face Recognition", collection of various methods, I-Tech Education and Publishing, Vienna - Austria, June 2007

[17] Turk, M., Pentland, A., "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, Vol. 3, pp. 71-86, 1991

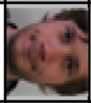
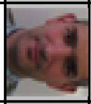
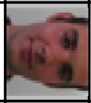
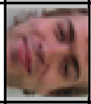
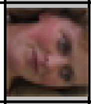
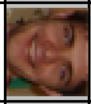
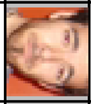
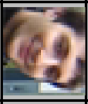
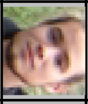
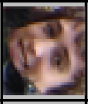
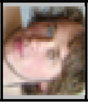
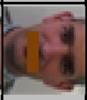
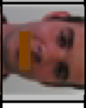
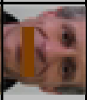
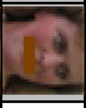
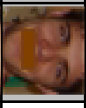
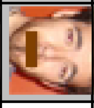
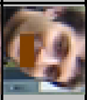
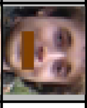
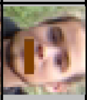
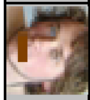
[18] Fritsch, L., "Metoda PCA a její implementace v jazyce C++", article, FEL České Vysoké Učení Technické v Praze

[19] Mathematics Source Library C & ASM, [cit. 2/6/2009], Available from WWW: http://mymathlib.webtrellis.net/index.html

[20] Jacobi's cyclic method, Mathematics Source Library C & ASM, [cit. 2/6/2009], Available from WWW: http://mymathlib.webtrellis.net/matrices/c_source/eigen/jacobi_cyclic_method.c

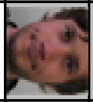
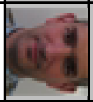
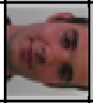
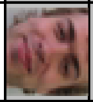
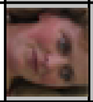
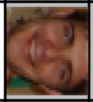
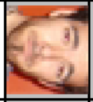
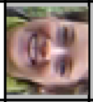
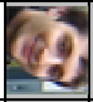
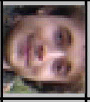
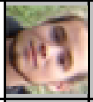
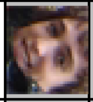
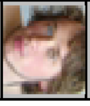
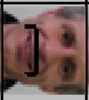
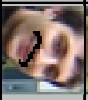
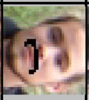
Appendix A1, Table:

Euclidian distances between Training set of faces in Eigenspace and Modified faces from Training set in Eigenspace. Modified face with covered mouth.

A1		Faces from Training set:														
		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
																
Modified faces from Training set:																
01_1		32.5	349.0	167.6	192.0	548.7	451.8	460.0	453.1	758.3	367.2	439.0	317.7	691.3	404.6	525.9
02_1		337.3	18.5	244.6	234.9	474.9	302.7	319.5	279.5	518.2	235.3	202.9	202.4	494.6	369.3	280.8
03_1		160.7	242.1	18.5	161.3	569.7	437.8	442.5	421.2	701.5	339.1	298.4	241.3	651.4	317.9	455.6
04_1		197.8	221.2	160.7	17.9	422.8	319.7	331.4	306.5	603.7	254.2	346.8	238.2	549.0	435.3	370.0
05_1		538.5	450.1	548.8	402.1	41.4	256.7	294.6	265.2	459.4	358.5	610.7	499.9	424.1	772.5	351.0
06_1		451.1	283.5	438.1	324.9	257.1	17.0	163.7	116.5	348.9	166.1	448.6	346.5	294.1	600.0	276.4
07_1		439.1	282.4	417.8	311.1	312.1	176.7	45.6	118.7	357.1	155.5	392.9	302.7	318.9	551.9	260.9
08_1		438.1	260.6	410.1	295.8	268.2	121.1	121.2	28.1	351.6	148.4	403.9	320.3	322.9	578.9	222.2
09_1		682.2	457.8	648.6	541.0	353.9	263.9	260.4	255.5	106.8	359.1	554.3	516.9	170.4	757.1	343.3
10_1		342.2	229.7	327.5	242.9	364.1	184.9	184.3	172.4	461.4	36.0	366.8	214.3	406.3	472.9	318.5
11_1		401.0	207.3	288.2	334.1	611.3	450.4	418.5	403.6	593.1	354.8	38.7	237.8	585.9	287.6	372.8
12_1		293.8	218.0	245.8	252.3	522.2	373.4	342.7	349.7	601.3	241.6	263.5	43.1	566.3	303.7	416.7
13_1		668.3	472.1	640.6	531.7	364.2	261.0	272.7	278.6	194.9	356.3	573.7	522.3	52.9	763.6	386.6
14_1		390.2	397.0	341.6	457.5	799.2	619.8	587.8	600.3	812.0	489.4	313.2	331.4	777.4	20.3	624.8
15_1		524.7	259.7	454.3	372.7	367.2	277.5	285.3	220.8	387.1	308.4	360.7	396.0	410.6	599.1	39.0

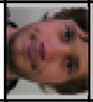
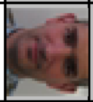
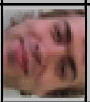
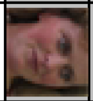
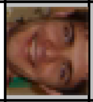
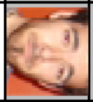
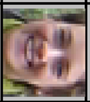
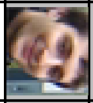
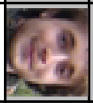
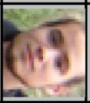
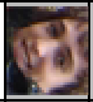
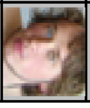
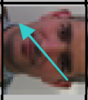
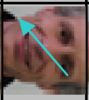
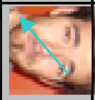
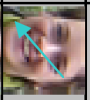
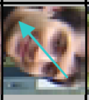
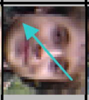
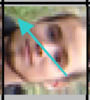
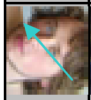
Appendix A2, Table:

Euclidian distances between Training set of faces in Eigenspace and Modified faces from Training set in Eigenspace. Modified face with added moustache (this modification is for testing purpose only, there is NO taunt, caricature, etc.)

A2		Faces from Training set:														
Modified faces from Training set:		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
																
01_2		175.6	268.9	226.5	**160.4**	420.9	296.2	312.0	312.3	603.3	221.7	393.2	236.4	534.8	425.1	424.7
02_2		319.9	54.1	225.8	225.7	487.7	329.0	337.0	299.1	552.0	253.4	209.5	206.0	530.2	358.3	292.2
03_2		219.2	180.7	117.1	**116.8**	489.1	357.3	365.1	333.3	610.9	280.6	275.8	242.1	560.8	384.7	362.5
04_2		261.9	206.5	228.5	85.1	356.1	255.4	274.0	239.2	538.9	216.3	353.9	253.3	486.6	481.1	308.4
05_2		695.0	591.5	707.1	558.3	152.7	355.0	383.9	371.4	444.2	482.6	740.0	644.4	414.1	918.5	447.1
06_2		377.9	231.2	347.8	245.2	327.4	141.1	231.0	165.1	435.6	165.1	394.9	314.1	364.4	547.3	274.1
07_2		395.9	275.7	377.3	268.8	327.0	202.8	102.5	142.8	408.1	153.1	375.6	278.5	362.3	527.1	275.3
08_2		423.4	286.9	394.1	274.8	290.7	177.3	192.9	132.4	412.7	193.0	437.9	354.8	357.5	605.0	262.8
09_2		823.9	614.4	802.7	684.3	393.2	395.1	394.3	395.2	184.7	509.8	713.9	670.3	265.3	918.6	465.2
10_2		338.8	240.4	318.4	232.4	374.0	199.8	206.9	188.2	475.5	80.9	384.6	239.0	405.6	490.0	334.9
11_2		415.9	175.2	304.4	317.6	559.4	398.2	361.6	344.5	520.7	318.4	75.9	246.7	514.8	350.5	315.9
12_2		275.9	211.7	189.2	219.0	551.2	397.3	377.1	366.5	631.2	263.0	254.0	115.7	583.7	312.3	418.7
13_2		672.4	457.2	632.1	534.0	411.0	286.9	291.9	291.1	196.0	367.8	545.4	523.4	63.2	746.8	373.5
14_2		341.2	297.2	282.0	373.9	696.0	511.8	482.5	490.8	709.7	382.0	235.4	232.8	673.0	113.5	521.4
15_2		632.1	383.3	564.2	467.2	372.1	336.9	333.9	273.0	351.0	398.3	467.1	511.1	390.6	722.4	126.4

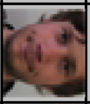
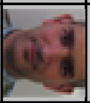
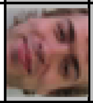
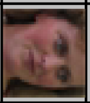
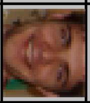
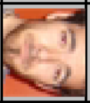
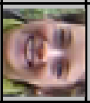
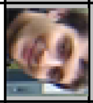
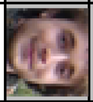
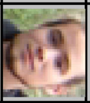
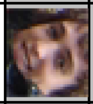
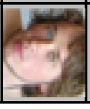
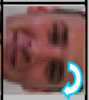
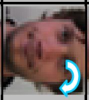
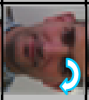
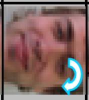
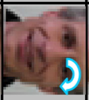
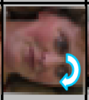
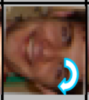
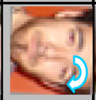
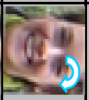
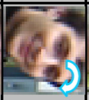
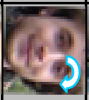
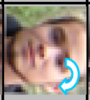
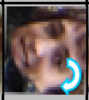
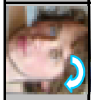
Appendix A3, Table:

Euclidian distances between Training set of faces in Eigenspace and Modified faces from Training set in Eigenspace. Modified face with shift of 1 pixel to left-down.

A3		Faces from Training set:														
Modified faces from Training set:		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
																
01_3		156.1	336.7	172.1	206.2	576.1	458.0	436.8	450.3	728.1	344.4	385.2	250.1	660.5	338.3	524.1
02_3		340.9	121.2	269.8	219.7	429.5	286.9	270.6	263.9	492.4	238.9	254.5	220.1	463.1	409.2	278.0
03_3		135.0	272.3	106.7	129.7	522.8	402.1	414.5	398.4	693.1	303.7	362.0	239.5	623.2	376.7	462.4
04_3		261.3	267.2	225.5	88.9	400.2	315.5	327.9	292.2	593.6	258.1	391.2	279.7	534.7	502.4	363.3
05_3		676.0	510.7	670.4	538.4	219.8	285.6	301.5	301.7	314.1	400.6	638.3	551.9	318.8	816.4	381.2
06_3		542.8	388.9	534.4	404.1	212.7	147.1	198.7	191.1	311.9	262.1	530.3	422.2	256.4	693.3	334.5
07_3		491.8	300.6	457.0	357.3	333.9	208.8	132.4	159.5	326.3	184.8	392.3	289.5	313.7	563.4	291.5
08_3		463.1	261.3	418.5	303.0	282.5	149.4	172.1	89.1	352.9	173.4	410.8	333.6	321.0	595.4	215.5
09_3		759.6	509.8	713.4	611.8	430.0	340.0	337.5	323.1	87.6	427.7	597.4	588.1	165.7	814.8	372.9
10_3		396.6	266.3	376.7	275.5	314.7	168.0	201.9	173.6	430.2	119.0	405.7	249.8	368.9	535.8	330.6
11_3		475.4	225.7	357.3	393.7	636.3	458.7	431.6	410.9	552.6	381.6	81.4	305.4	558.6	342.0	363.4
12_3		278.1	176.4	219.9	225.1	505.4	343.6	313.0	314.8	563.0	225.0	211.8	97.0	526.4	299.7	376.9
13_3		710.9	491.3	663.2	569.8	458.6	330.9	327.1	323.8	181.4	394.4	562.2	546.2	102.1	767.9	404.5
14_3		458.4	428.2	389.0	509.5	852.4	673.1	637.2	645.8	841.6	546.9	313.7	383.7	820.7	101.7	645.8
15_3		495.8	244.4	433.7	331.8	301.9	224.5	235.8	170.4	361.5	275.8	377.2	375.9	368.0	604.0	93.3

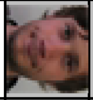
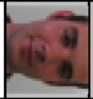
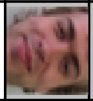
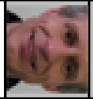
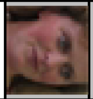
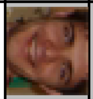
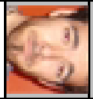
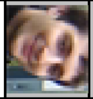
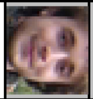
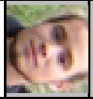
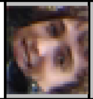
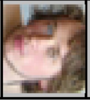
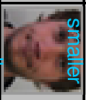
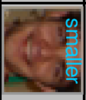
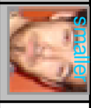
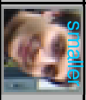
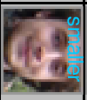
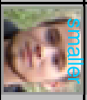
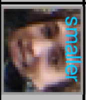
Appendix A4, Table:

Euclidian distances between Training set of faces in Eigenspace and Modified faces from Training set in Eigenspace. Modified face with rotation of 3 degree in the direction of clock.

A4		Faces from Training set:														
		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
																
Modified faces from Training set: 01_4		201.8	261.6	236.6	**138.6**	381.3	267.0	301.8	278.2	584.9	217.8	406.0	269.8	520.4	472.9	388.1
02_4		448.8	167.4	387.0	314.6	373.0	209.3	235.0	182.1	369.2	225.5	311.0	299.2	377.4	508.7	203.7
03_4		164.8	246.6	37.8	171.9	583.0	449.7	452.9	432.5	712.2	354.0	294.3	251.9	664.3	309.4	457.9
04_4		291.9	200.3	263.0	150.6	332.3	225.2	260.2	223.3	508.2	222.1	361.3	290.9	452.1	502.0	279.2
05_4		690.1	579.5	708.1	566.8	167.4	322.1	370.7	355.5	422.2	449.7	736.7	620.0	391.9	895.5	465.7
06_4		531.2	357.4	517.8	399.9	251.7	92.3	188.1	142.6	282.2	231.8	505.6	418.5	232.5	673.0	307.9
07_4		416.2	280.9	406.8	309.5	324.4	180.9	72.9	130.1	380.5	140.8	389.3	283.7	341.9	529.2	291.9
08_4		406.0	233.4	369.3	264.9	328.3	154.7	142.5	88.0	377.5	121.1	364.2	275.1	342.8	528.3	254.4
09_4		749.3	507.2	705.2	606.3	438.9	341.0	333.7	327.1	50.7	431.9	587.6	582.6	159.2	801.9	388.5
10_4		390.1	237.6	371.4	283.7	353.2	149.9	144.7	140.0	400.6	68.4	365.3	245.6	342.6	492.8	300.5
11_4		458.7	234.3	342.0	374.9	625.4	463.3	419.8	409.2	562.4	380.6	69.1	297.6	561.8	341.0	358.2
12_4		328.8	191.4	275.0	257.4	485.2	315.0	283.9	287.6	527.7	179.8	253.6	66.6	494.3	346.0	371.4
13_4		754.6	543.1	724.1	621.1	432.9	347.7	342.7	358.5	161.9	442.8	628.8	599.9	79.1	825.3	439.4
14_4		403.5	348.0	357.6	442.3	735.2	549.7	509.4	528.7	718.0	429.1	262.1	294.1	692.1	120.0	554.4
15_4		530.0	267.0	463.2	375.1	335.1	253.0	275.9	195.3	368.8	294.9	385.6	400.2	390.0	618.6	65.9

Appendix A5, Table:

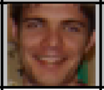
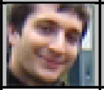
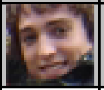
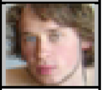
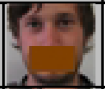
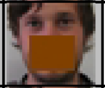
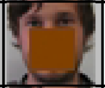
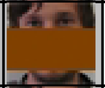
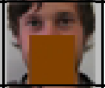
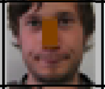
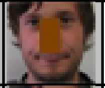
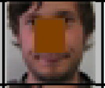
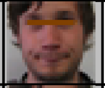
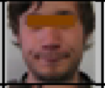
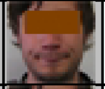
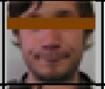
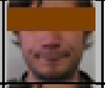
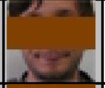
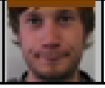
Euclidian distances between Training set of faces in Eigenspace and Modified faces from Training set in Eigenspace. Modified face - smaller size of faces (95% in x and 95% in y).

A5		Faces from Training set:														
Modified faces from Training set:		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
																
01_5		226.1	347.7	273.1	**166.3**	406.8	331.5	349.6	346.2	641.6	276.6	482.1	313.5	554.2	524.2	465.6
02_5		501.9	312.0	468.2	341.1	**204.7**	**146.1**	**202.3**	**152.7**	341.8	**250.6**	466.2	398.2	**296.0**	659.5	**216.8**
03_5		201.1	283.2	194.1	**105.2**	432.9	341.4	373.8	343.7	648.5	275.5	413.6	275.5	575.5	476.9	419.3
04_5		329.8	342.6	322.8	179.7	325.7	307.0	344.2	302.0	591.8	301.3	483.0	362.8	522.8	604.3	379.5
05_5		659.2	563.7	671.8	523.7	139.1	324.9	362.0	349.0	453.4	444.1	718.1	610.1	403.4	885.8	431.0
06_5		624.8	491.1	628.6	492.4	**180.3**	223.6	286.7	266.2	347.4	364.9	641.8	549.1	280.2	814.6	374.3
07_5		479.8	439.8	510.1	374.2	**223.2**	**253.6**	257.3	271.5	472.9	328.8	594.0	484.8	394.8	719.2	405.9
08_5		489.0	440.4	505.9	366.1	**197.3**	**267.5**	310.5	282.7	491.2	364.2	601.6	514.3	422.9	751.2	379.2
09_5		713.4	476.6	671.3	563.0	368.2	309.5	306.8	295.0	133.7	421.1	575.7	576.1	184.0	799.1	323.3
10_5		337.0	215.2	311.6	209.6	335.3	179.0	191.8	161.0	462.0	103.8	358.0	238.8	401.8	490.6	279.7
11_5		379.9	**119.2**	283.5	265.2	477.4	326.3	327.3	285.3	489.9	284.5	181.0	268.9	480.6	407.8	261.8
12_5		293.2	233.5	284.4	**173.1**	337.8	**219.3**	231.2	**209.0**	511.1	**143.8**	385.1	219.3	451.1	488.5	345.8
13_5		682.1	471.3	647.7	541.3	363.0	289.6	284.7	283.6	194.2	389.4	564.6	548.1	122.0	775.9	342.4
14_5		261.5	**221.1**	**210.5**	281.3	598.7	431.5	413.0	416.8	656.1	330.3	233.3	242.6	615.9	221.3	437.1
15_5		557.7	317.5	505.8	402.4	287.2	217.5	245.8	182.1	299.9	305.5	435.8	438.7	311.5	665.3	125.0

Appendix A6_1, Table:

Euclidian distances between Training set of faces in Eigenspace and Modified faces from Training set in Eigenspace.

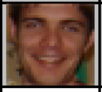
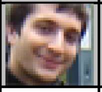
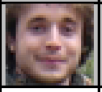
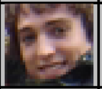
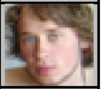
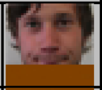
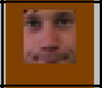
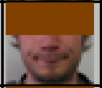
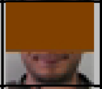
Modified face 02 with some changes (testing one face 02 with modifies).

A6_1			Faces from Training set:														
			01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
																	
Modified face 02 from Training set:	02_6_1		321.1	40.2	224.6	219.2	480.0	317.5	333.3	291.9	543.5	245.5	210.0	204.2	518.8	366.2	286.1
	02_6_2		278.3	111.7	164.1	203.3	529.8	369.7	382.3	347.6	609.3	278.3	219.2	201.3	572.1	327.5	350.3
	02_6_3		261.0	129.4	149.9	195.1	532.9	370.2	386.0	352.8	619.3	273.6	235.2	194.8	575.2	323.0	368.6
	02_6_4		**210.6**	338.8	**137.1**	**272.4**	673.9	554.5	551.5	531.3	820.6	446.0	364.8	**330.8**	770.1	**319.6**	544.1
	02_6_5		276.8	97.4	175.2	175.7	485.1	333.3	349.4	311.1	583.9	247.9	240.9	192.2	546.8	362.2	319.2
	02_6_6		308.2	47.6	211.8	195.5	463.5	301.9	320.0	277.3	534.8	231.4	225.1	201.5	504.0	379.8	289.3
	02_6_7		313.8	41.2	216.1	206.3	472.7	308.0	320.7	280.8	534.7	233.5	213.2	198.6	506.0	370.8	287.9
	02_6_8		308.2	74.2	200.9	217.9	506.6	340.8	354.9	317.8	566.8	260.7	212.0	203.6	534.1	351.8	323.0
	02_6_9		348.8	18.8	257.0	243.0	469.5	295.1	310.8	270.0	504.4	231.0	208.1	208.4	482.7	380.0	273.3
	02_6_10		364.8	29.5	269.8	253.9	472.6	296.3	310.8	269.4	494.9	236.3	204.1	218.6	475.4	388.2	264.4
	02_6_11		338.9	36.3	249.2	234.6	470.6	288.0	308.0	269.8	505.1	217.0	214.6	191.6	476.8	373.1	291.6
	02_6_12		376.4	50.0	276.4	261.9	479.9	305.7	319.2	278.0	493.5	253.8	199.6	235.4	473.0	395.5	260.7
	02_6_13		375.7	59.5	279.9	250.1	458.1	282.1	299.6	256.0	474.7	232.3	206.7	218.9	450.0	407.1	254.5
	02_6_14		268.2	262.2	**145.9**	274.1	658.0	506.7	504.3	483.9	737.4	407.4	272.8	296.4	690.9	280.6	486.2
	02_6_15		430.8	138.9	365.0	297.7	380.3	190.1	228.1	172.6	379.1	189.7	292.6	271.0	360.1	489.3	205.1

Appendix A6_2, Table:

Euclidian distances between Training set of faces in Eigenspace and Modified faces from Training set in Eigenspace.

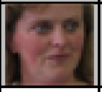
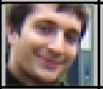
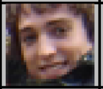
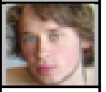
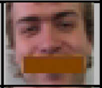
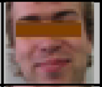
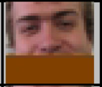
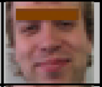
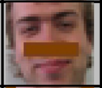
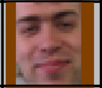
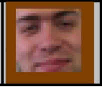
Modified face 02 with some changes (testing one face 02 with modifies) - continuation.

A6_2			Faces from Training set:														
			01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
																	
Modified face 02 from Training set:	02_6_16		267.7	127.5	158.7	201.2	528.9	381.5	384.8	350.5	618.5	287.2	210.9	200.6	591.4	315.6	348.5
	02_6_17		303.9	110.7	219.4	177.1	438.4	285.2	278.1	243.0	516.3	194.0	231.5	173.2	485.6	392.0	272.0
	02_6_18		519.0	290.5	475.3	369.8	294.5	**167.5**	**151.8**	**115.6**	**268.0**	**237.7**	408.9	383.0	**258.7**	616.6	**178.5**
	02_6_19		482.1	186.5	410.0	345.9	403.5	209.6	243.8	191.8	336.6	224.8	302.6	307.2	319.3	518.7	217.6
	02_6_20		452.7	172.2	387.8	318.4	383.1	187.4	225.5	**173.0**	350.9	198.4	300.5	285.1	329.8	504.0	218.8
	02_6_21		297.9	133.8	197.8	202.6	503.5	329.7	338.6	310.7	551.3	253.0	230.1	221.4	497.9	366.5	343.3
	02_6_22		327.0	102.4	239.6	231.5	479.4	322.0	323.7	292.5	547.8	230.5	235.0	190.7	521.8	378.2	300.3
	02_6_23		297.4	76.5	205.8	216.8	494.7	328.5	343.2	309.8	563.8	249.8	232.1	196.6	536.1	351.2	325.9
	02_6_24		275.0	101.3	183.7	214.1	516.5	352.4	369.8	335.2	594.9	267.9	229.5	209.3	566.1	323.3	345.1
	02_6_25		271.1	209.0	**149.5**	236.0	601.8	457.1	459.6	427.4	687.0	357.7	236.0	223.0	665.6	288.4	434.9
	02_6_26		284.4	181.7	254.3	191.1	400.0	281.1	312.5	277.0	568.9	215.2	343.7	201.2	537.5	431.1	329.9
	02_6_27		581.8	569.3	**480.5**	622.8	1004.4	838.8	825.1	806.5	1005.6	733.0	**467.3**	594.9	978.0	**392.2**	770.6
	02_6_28		358.1	27.3	263.8	250.6	475.5	296.9	313.1	273.7	498.6	235.4	202.8	214.1	476.2	380.7	274.2
	02_6_29		359.3	32.4	260.2	247.9	478.1	303.1	322.2	276.4	504.4	245.0	203.3	225.8	482.7	387.8	269.7
	02_6_30		262.6	229.7	242.9	**198.7**	430.3	337.3	374.5	333.0	632.7	271.3	377.5	238.3	599.7	439.2	377.4

Appendix A7, Table:

Euclidian distances between Training set of faces in Eigenspace and Modified faces from Training set in Eigenspace.

Modified face 05 with some changes (testing one face 05 with modifies).

A7			Faces from Training set:														
			01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
																	
Modified face 02 from Training set:	05_7_1		533.6	453.1	549.4	404.1	56.0	252.1	302.7	271.0	468.7	357.3	615.9	500.4	431.2	771.5	363.4
	05_7_2		529.1	467.9	551.3	405.8	68.0	270.7	310.0	289.2	487.6	366.3	628.3	504.1	442.7	774.5	390.2
	05_7_3		357.9	260.2	362.9	**243.1**	255.2	**146.2**	**165.9**	**163.0**	437.8	**166.4**	410.4	286.9	382.6	540.1	296.2
	05_7_4		497.1	375.5	494.8	356.2	101.3	212.7	245.9	212.6	421.8	309.9	531.9	437.4	402.1	700.7	283.8
	05_7_5		465.6	417.2	484.8	341.0	118.5	248.6	298.2	265.8	504.4	328.5	580.7	459.2	451.3	724.3	363.8
	05_7_6		472.0	367.0	462.2	323.1	159.8	212.4	241.0	195.8	445.3	263.7	520.5	403.2	396.9	685.2	300.7
	05_7_7		**325.7**	**124.6**	**268.5**	**214.1**	401.3	**257.3**	**247.1**	**227.7**	488.2	**172.5**	**257.7**	**158.9**	463.5	**401.2**	**279.1**

Appendix B1 – Project diagram

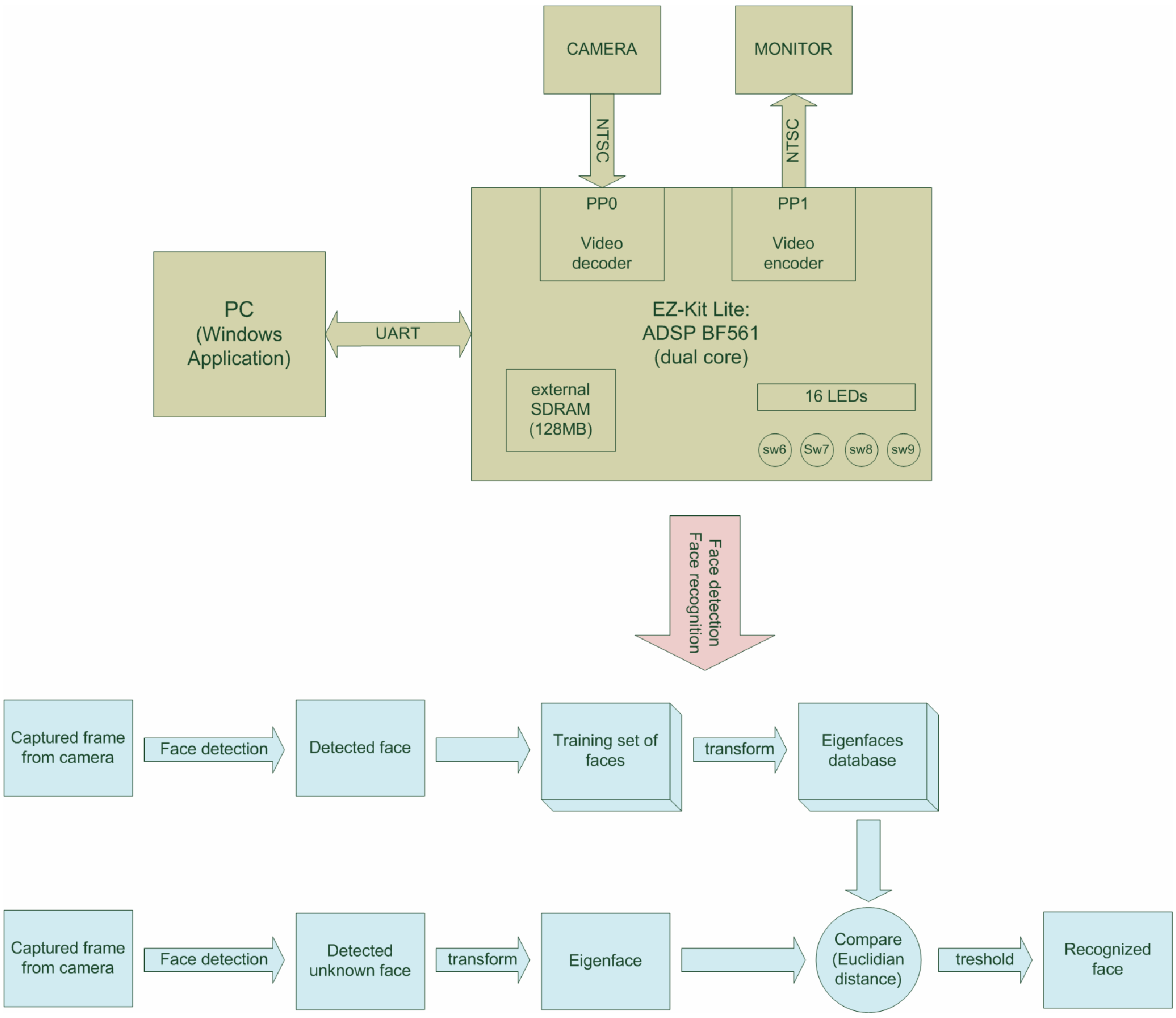


Figure 22: *Project diagram: Development board with connections to PC, camera, monitor, and implemented Face detection and Face recognition algorithms into DSP processor.*

Appendix C1 – Source code of project for DSP processor

This section contents source code of project for DSP processor. Source code is divided into some files, in appendix are shown these files from source code:

§ Main C file – impl. alg.

§ Memory L3, C file

§ Header1, C file

The Main C file contents all implemented algorithms without libraries (DSP and run-time library for DSP processor) and extern function such as "YCbCrtoRGB.asm" and "jacobi_cyclic_method.c" which are not designed by author.

```
/*********************************************************************************

Copyright(c) 2005 Analog Devices, Inc. All Rights Reserved.
Copyright(c) 2009 Peter KNAPO, (PK). All Rights Reserved.

*** Project: 2_face-detect_send-PC ***
***  + implementation of Eigenfaces ***
*** for DSP: ADSP BF561 in EZ-Kit Lite ***

$Revision: 1.2 $
$Date: 2007/05/18 20:35:02 $

PK Revision: 4.2
PK Date:     1. August 2009, 15:26

*********************************************************************************/


/*********************************************************************

Include files

*********************************************************************/

#include <services\services.h>                          // system services
#include <drivers\adi_dev.h>                            // device manager includes
#include <drivers\ppi\adi_ppi.h>                        // PPI driver includes
#include <drivers\uart\adi_uart.h>                      // uart driver includes

#include "ezkitutilities.h"                                     // EZ-Kit utilities
#include "adi_itu656.h"                             // ITU656 utilities

#include "../Header1.h"                                         // Header file for project ( my
#define )

#include <math.h>                                                       // DSP RUN-TIME LIBRARY
#include <filter.h>                                             // DSP RUN-TIME LIBRARY
#include <matrix.h>                                                     // DSP RUN-TIME LIBRARY
#include <stdlib.h>                                                     // Run-time library

#include "jacobi_cyclic_method.c"                       // Jacobi cyclic method for counting
eigenvalues and eigenvectors for symetric matrix (covariance matrix)



/*********************************************************************

ADSP-BF533/537 have only 1 PPI called PPI0
ADSP-BF561 has 2, PPI0 connected to video decoder, PPI1 to video encoder

*********************************************************************/

#if defined(__ADSPBF561__)
#define ENCODER_PPI (1)
#define DECODER_PPI (0)
#else
#define ENCODER_PPI (0)
#define DECODER_PPI (0)
#endif



/*********************************************************************

User configurations:

Callbacks can be either "live" meaning they happen at hardware interrupt
time, or "deferred" meaning that the Deferred Callback Service is used
to make callbacks at a lower priority interrupt level.  Deferred
callbacks usually allow the system to process data more efficiently with
lower interrupt latencies.

The macro below can be used to toggle between "live" and "deferred"
callbacks.  All drivers and system services that make callbacks into an
application are passed a handle to a callback service.  If that handle
is NULL, then live callbacks are used.  If that handle is non-NULL,
meaning the handle is a real handle into a deferred callback service,
then callbacks are deferred using the given callback service.

*********************************************************************/


/*********************************************************************

Static and Extern data

*********************************************************************/


// DMA Manager data (base memory + memory for 1 DMA channel)
static u8 DMAMgrData[ADI_DMA_BASE_MEMORY + (ADI_DMA_CHANNEL_MEMORY * 1)]; //x1

// Deferred Callback Manager data (memory for 1 service plus 4 posted callbacks)
#if defined(USE_DEFERRED_CALLBACKS)
static u8 DCBMgrData[ADI_DCB_QUEUE_SIZE + (ADI_DCB_ENTRY_SIZE)*4];
#endif
```

```
// Device Manager data (base memory + memory for 1 device)
static u8 DevMgrData[ADI_DEV_BASE_MEMORY + (ADI_DEV_DEVICE_MEMORY * 1)];

// define a chunk of SDRAM to hold a single NTSC video frame
//      For single-core processors, we can define the frame right here but
//      for dual-core processors, we need to declare it here but define
//      it in the sml3 project so that they get placed in SDRAM properly

#if defined(__ADSPBF561__)
        //extern u8 Frame[];
        //extern u8 Frame_A[]; // 720 x 525
  extern u8 Frame_ycbcr[]; // 1440 x 525 = 756000, in External memory SDRAM - L3
  extern u8 rgb_reduced_frame[]; // R G B ... 3x180x120 = 64800 , in L2 memory: R1 G1 B1 R2 G2 B2
  extern u8 byte_array_reduced[]; // 180x120, in L2 memory
  extern u8 Y_reduced_frame[]; // 180x120, in L2 memory: Y1 Y2 Y3
  extern u8 Skin_noise_remove[]; // 36x24,  0/255 0/255 ...
  extern u8 Skin_noise_remove_2D[Y_R_REM_FRAME][X_R_REM_FRAME]; // 36x24, [Y][X], in L2 memory
  extern double Face1_Y[MAX_FACES_IN_MEMORY][FACE_DET_SIZE]; // Detected faces in grayscale from
Y_reduced_frame[]
  extern u8 Face_one_Y_u8[FACE_DET_SIZE];
  extern double One_face[1][FACE_DET_SIZE];
  extern double Face1_Y_transpose[FACE_DET_SIZE][MAX_FACES_IN_MEMORY]; // Detected faces in
grayscale from Y_reduced_frame[] - transposed
  extern double Mean_face[FACE_DET_SIZE]; // Mean face from Face1_Y[]
  extern double Covariance_matrix_m[FACE_DET_SIZE][FACE_DET_SIZE];
  extern double Eigenvalues[MAX_FACES_IN_MEMORY];
  extern double Eigenvectors_m[MAX_FACES_IN_MEMORY][MAX_FACES_IN_MEMORY];
  extern double Eigenvectors[MAX_FACES_IN_MEMORY][FACE_DET_SIZE];
  extern double P_Eigenvalues[MAX_FACES_IN_MEMORY];
  extern double Eigenvectors_transpose[FACE_DET_SIZE][MAX_FACES_IN_MEMORY];
  extern double Eigenface_database[MAX_FACES_IN_MEMORY][MAX_FACES_IN_MEMORY];
  extern double One_eigenface[1][MAX_FACES_IN_MEMORY];
  extern double Distance_recognition[MAX_FACES_IN_MEMORY];
  extern u8 Distance_recognition_u8[MAX_FACES_IN_MEMORY];
  extern u8 Array_u8[FACE_DET_SIZE];
     //povodne//extern u8 small_array_2D[Y_R_REM_FRAME][X_R_REM_FRAME]; // 36x24, in L2 memory
     //povodne//extern fract16 out_conv_array[Y_R_REM_FRAME+A_CONV_Y-1][X_R_REM_FRAME+A_CONV_X-1];
// in L2 memory
#else
        //static u8 Frame[ADI_ITU656_NTSC_LINE_WIDTH * ADI_ITU656_NTSC_HEIGHT];
        //static u8 Frame_A[ADI_ITU656_NTSC_WIDTH * ADI_ITU656_NTSC_HEIGHT]; // 720 x 525
 //static u8 Frame_ycbcr[1440 * ADI_ITU656_NTSC_HEIGHT]; // 1440 x 525
 //static u8 rgb_reduced_frame[X_R_FRAME*Y_R_FRAME][3]; // R G B ... 3x180x120 = 64800 , in L1
memory: R1 G1 B1 R2 G2 B2
#endif


// Buffers for UART:
ADI_DEV_1D_BUFFER OutboundBuffer;
ADI_DEV_1D_BUFFER OutboundBuffer2;
ADI_DEV_1D_BUFFER OutboundBuffer3;
ADI_DEV_1D_BUFFER InboundBuffer;

u8 OutboundData[]="abcdefghijklmnop";

// Handlers:
ADI_DCB_HANDLE                  DCBManagerHandle;               // handle to the callback service
ADI_DMA_MANAGER_HANDLE  DMAManagerHandle;               // handle to the DMA Manager
ADI_DEV_MANAGER_HANDLE  DeviceManagerHandle;    // handle to the Device Manager
//ADI_DEV_DEVICE_HANDLE         DriverHandle_uart; // handle to the uart driver

u8 uart_processed1=0; // processed flag for uart

// function: Convert YCbCr to RGB from .asm file: YCbCrtoRGB.asm
void (*f1)();
void _YCbCrtoRGB();

extern void Create_eigenface_database(void);

  //povodne// Arrays in L1 Memory:
  /*u8 array_conv[A_CONV_Y][A_CONV_X]=
  {
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,1}
  };*/
  //povodne// u8 Skin_noise_remove_2D[Y_R_REM_FRAME][X_R_REM_FRAME]; // 36x24, in L1 memory

unsigned int num_faces=0; // pointer of detected face to create Eigenface database (i-th detected
faces in memory)
unsigned int i1_test=0;
unsigned int ig1=0;



/*********************************************************************

        Function:               ExceptionHandler
                                        HWErrorHandler

        Description:    We should never get an exception or hardware error,
                                        but just in case we'll catch them and simply turn
                                        on all the LEDS should one ever occur.
```

```
*********************************************************************/

static ADI_INT_HANDLER(ExceptionHandler)        // exception handler
{
                ezErrorCheck(1);
                return(ADI_INT_RESULT_PROCESSED);
}


static ADI_INT_HANDLER(HWErrorHandler)          // hardware error handler
{
                ezErrorCheck(1);
                return(ADI_INT_RESULT_PROCESSED);
}


/*********************************************************************

        Function:               Callback

        Description:    Each type of callback event has it's own unique ID
                                so we can use a single callback function for all
                                callback events.  The switch statement tells us
                                which event has occurred.

                                In the example, we'll get a callback when the PPI
                                has completed processing of the input buffer.  We
                                don't really need the callback function as the main
                                code uses the ProcessedFlag field of the buffer to
                                determine when the inbound buffer has completed
                                processing, but it's included here for illustrative
                                purposes.

                                Note that in the device driver model, in order to
                                generate a callback for buffer completion, the
                                CallbackParameter of the buffer must be set to a non-NULL
                                value.  That non-NULL value is then passed to the
                                callback function as the pArg parameter.  In the example,
                                the CallbackParameter is set to be the address of the
                                buffer itself.  That allows     the callback function to
                                determine which buffer was processed.  Often the
                                CallbackParameter value is set to NULL for every buffer
                                except the last buffer in a chain. That technique causes
                                the callback function to get called only when the last
                                buffer in the chain has been processed.

*********************************************************************/


static void Callback(
        void *AppHandle,
        u32  Event,
        void *pArg)
{

        static unsigned int Counter = 0;        // count the number of input buffers processed
        ADI_DEV_BUFFER *pBuffer;                        // pointer to the buffer that was processed

        // CASEOF (event type)
        switch (Event) {

                // CASE (buffer processed)
                case ADI_DEV_EVENT_BUFFER_PROCESSED:

                        // point to the buffer
                        pBuffer = (ADI_DEV_BUFFER *)pArg;

                        // increment our counter
                        Counter++;

                        if (pBuffer == (ADI_DEV_BUFFER *)&OutboundBuffer) // if processed
OutboundBuffer - UART
                        {
                                uart_processed1=1; // UART was processed
                        }

                        break;

                // CASE (an error)
                case ADI_DEV_EVENT_DMA_ERROR_INTERRUPT:
                case ADI_PPI_EVENT_ERROR_INTERRUPT:

                        // turn on all LEDs and wait for help
                        ezTurnOnAllLEDs();
                        while (1) ;

        // ENDCASE
        }

        // return
}


/*********************************************************************

        Function:               CaptureFrame
        Description:    This function is called to capture a frame of video
                                data.  This function first enables the AD7183 video
                                decoder and gives it time to sync.  The buffer that
                                is passed to the device driver, given to this function
                                by the calling function, is modified so as to generate
                                a callback when the buffer is processed by the PPI
                                driver.  The PPI driver is then opened for input,
                                configured according to the parameters in the
                                configuration table, and then dataflow is enabled.
                                After the buffer has been processed, meaning the
                                frame of video data has been captured in SDRAM, the
                                PPI driver is closed and the function returns to the
                                caller.

                                The first LED is lit when this function is executing.

*********************************************************************/

void CaptureFrame(
        ADI_DCB_HANDLE                  DCBManagerHandle,
        ADI_DMA_MANAGER_HANDLE  DMAManagerHandle,
        ADI_DEV_MANAGER_HANDLE  DeviceManagerHandle,
        ADI_DEV_2D_BUFFER               *pBuffer2D
){

        // table of configuration values for the PPI on input
        ADI_DEV_CMD_VALUE_PAIR InboundConfigurationTable [] = {
                { ADI_DEV_CMD_SET_DATAFLOW_METHOD,              (void *)ADI_DEV_MODE_CHAINED    },
                { ADI_PPI_CMD_SET_CONTROL_REG,                  (void *)0x00c0
        }, // Avtive field only, Field 1 & 2, YcbCr - all //  ( 0x02c0   0x0084 )
                { ADI_PPI_CMD_SET_LINES_PER_FRAME_REG,  (void *)LINES_FROM_FRAME_NTSC   }, //
ADI_ITU656_NTSC_HEIGHT
                { ADI_DEV_CMD_END,                                                      NULL
                        },
        };

        ADI_DEV_DEVICE_HANDLE   DeviceHandle;   // handle to the device driver

        // turn on first LED
        ezTurnOnLED(0);

        // enable the video decoder (7183)
        ezEnableVideoDecoder();

        // give the decoder time to sync
        ezDelay(300);

        // since we're doing input, have the PPI driver generate a callback when the buffer
        // has completed processing.  The pArg parameter to the callback function will be
        // whatever we set the CallbackParameter value to.
        pBuffer2D->CallbackParameter = pBuffer2D;

        // open the PPI driver for input
        ezErrorCheck(adi_dev_Open(DeviceManagerHandle, &ADIPPIEntryPoint, DECODER_PPI, NULL,
&DeviceHandle, ADI_DEV_DIRECTION_INBOUND, DMAManagerHandle, DCBManagerHandle, Callback));

        // configure the PPI driver with the values from the inbound configuration table
    ezErrorCheck(adi_dev_Control(DeviceHandle, ADI_DEV_CMD_TABLE, InboundConfigurationTable));

        // give the PPI driver the buffer to process
        ezErrorCheck(adi_dev_Read(DeviceHandle, ADI_DEV_2D, (ADI_DEV_BUFFER *)pBuffer2D));

        // tell the PPI driver to enable data flow
        ezErrorCheck(adi_dev_Control(DeviceHandle, ADI_DEV_CMD_SET_DATAFLOW, (void *)TRUE));

        // wait till the buffer has been processed
        while (pBuffer2D->ProcessedFlag == FALSE) ;

        // delay
        ezDelay(300);

        // close the PPI driver
        ezErrorCheck(adi_dev_Close(DeviceHandle));

        // turn off first LED
        ezTurnOffLED(0);

        // return
}




/*********************************************************************

        Function:               DisplayFrame

        Description:    This function is called to display a frame of video
                                data on the video-out device.  The function first
                                enables the AD7171 video encoder and gives it time
                                to sync.  As we're going to display the data for a
                                second or two, this function opens the PPI driver
                                using the chained with loopback mode.  This will
                                cause the PPI driver to continuously send the buffer
                                out the PPI so that we can see the image.  As a result
                                we don't need to be notified by a callback that the
```

```
                                        buffer has been processed, so the buffer that is sent
                                        to the PPI driver is modified to not generate a
                                        callback.  We then open the PPI driver and configured
                                        according to the parameters in the      configuration
                                        table, and then dataflow is enabled.  The function
                                        spins for a while, allowing the data to be seen on
                                        the video monitor, then closes down the PPI driver.

                                        The last LED is lit when this function is executing.

*********************************************************************/

void DisplayFrame(
        ADI_DCB_HANDLE                  DCBManagerHandle,
        ADI_DMA_MANAGER_HANDLE  DMAManagerHandle,
        ADI_DEV_MANAGER_HANDLE  DeviceManagerHandle,
        ADI_DEV_2D_BUFFER               *pBuffer2D
){

                // table of configuration values for the PPI on output
        ADI_DEV_CMD_VALUE_PAIR OutboundConfigurationTable [] = {
                { ADI_DEV_CMD_SET_DATAFLOW_METHOD,                      (void
*)ADI_DEV_MODE_CHAINED_LOOPBACK },
                { ADI_PPI_CMD_SET_CONTROL_REG,                          (void *)0x0082
                                },
                { ADI_PPI_CMD_SET_LINES_PER_FRAME_REG,          (void *)ADI_ITU656_NTSC_HEIGHT
},
                { ADI_DEV_CMD_END,                                                      NULL
                                                },
        };

        ADI_DEV_DEVICE_HANDLE   DeviceHandle;   // handle to the device driver

        // turn on 2. LED
        ezTurnOnLED(1);

        // enable video encoder (7171)
        ezEnableVideoEncoder();

        // give the decoder time to sync
        ezDelay(300);

        // since we're doing output, don't generate any callbacks
        pBuffer2D->CallbackParameter = NULL;

        // open the PPI driver for output
        ezErrorCheck(adi_dev_Open(DeviceManagerHandle, &ADIPPIEntryPoint, ENCODER_PPI, NULL,
&DeviceHandle, ADI_DEV_DIRECTION_OUTBOUND, DMAManagerHandle, DCBManagerHandle, Callback));

        // configure the PPI driver with the values from the outbound configuration table
    ezErrorCheck(adi_dev_Control(DeviceHandle, ADI_DEV_CMD_TABLE, OutboundConfigurationTable));

        // give the PPI driver the buffer to process
        ezErrorCheck(adi_dev_Write(DeviceHandle, ADI_DEV_2D, (ADI_DEV_BUFFER *)pBuffer2D));

        // tell the PPI driver to enable dataflow
        ezErrorCheck(adi_dev_Control(DeviceHandle, ADI_DEV_CMD_SET_DATAFLOW, (void *)TRUE));

                // delay for a while so we can see the image
                ezDelay(3000);

        // close the PPI driver
        ezErrorCheck(adi_dev_Close(DeviceHandle));

        // turn off 2. LED
        ezTurnOffLED(1);

        // return
}

/*********************************************************************

        Function:               Init_and_Send_buffer_to_UART

*********************************************************************/

void Init_and_Send_buffer_to_UART(
        ADI_DCB_HANDLE                  DCBManagerHandle_u,
        ADI_DMA_MANAGER_HANDLE  DMAManagerHandle_u,
        ADI_DEV_MANAGER_HANDLE  DeviceManagerHandle_u,
        ADI_DEV_1D_BUFFER               *pBuffer1D
)
{
        // table of configuration values for the UART
        ADI_DEV_CMD_VALUE_PAIR ConfigurationTable [] = {        // configuration table for the UART
driver
                { ADI_DEV_CMD_SET_DATAFLOW_METHOD,      (void *)ADI_DEV_MODE_CHAINED    },
                { ADI_UART_CMD_SET_DATA_BITS,           (void *)8
        },
                { ADI_UART_CMD_ENABLE_PARITY,           (void *)FALSE
},
                { ADI_UART_CMD_SET_STOP_BITS,           (void *)1
        },
                { ADI_UART_CMD_SET_BAUD_RATE,           (void *)BAUD_RATE
},
                { ADI_UART_CMD_SET_LINE_STATUS_EVENTS,  (void *)TRUE
},
                {  ADI_DEV_CMD_END,                                                     NULL
                        },
        };

        // Handle to the UART driver
        ADI_DEV_DEVICE_HANDLE   DriverHandle_u;

        //32 sended_bytes_uart=0;

        // turn on 3. LED
        ezTurnOnLED(2);

        // callback after buffer processed enabled
    pBuffer1D->CallbackParameter = pBuffer1D; //NULL; //pBuffer1D;

    //
    uart_processed1=0;

        // open the UART driver for bidirectional data flow
        ezErrorCheck(adi_dev_Open(DeviceManagerHandle_u, &ADIUARTEntryPoint, 0, NULL,
&DriverHandle_u, ADI_DEV_DIRECTION_BIDIRECTIONAL, NULL, DCBManagerHandle_u, Callback));

        // configure the UART driver with the values from the configuration table
        ezErrorCheck(adi_dev_Control(DriverHandle_u, ADI_DEV_CMD_TABLE, ConfigurationTable));


        // send the outbound buffer 1D
        ezErrorCheck(adi_dev_Write(DriverHandle_u, ADI_DEV_1D, (ADI_DEV_BUFFER *)pBuffer1D));

        // enable data flow
        ezErrorCheck(adi_dev_Control(DriverHandle_u, ADI_DEV_CMD_SET_DATAFLOW, (void *)TRUE));

                //ezErrorCheck(adi_dev_Control(DriverHandle_u,
ADI_UART_CMD_COMPLETE_OUTBOUND_BUFFER, NULL));
                //ezErrorCheck(adi_dev_Control(DriverHandle_u,
ADI_UART_CMD_GET_OUTBOUND_PROCESSED_ELEMENT_COUNT, &sended_bytes_uart));
                //printf(" ADI_UART_CMD_GET_OUTBOUND_PROCESSED_ELEMENT_COUNT = %d\n",
sended_bytes_uart);

        // wait till the buffer has been processed
        while (pBuffer1D->ProcessedFlag == FALSE) ;
        //while (uart_processed1==0) ;

        uart_processed1=0;

        // wait for sending last byte before close UART driver
        ezDelay(300);

        // close the UART driver
        ezErrorCheck(adi_dev_Close(DriverHandle_u));


        ezDelay(300); // delay for LED

        // turn off 3. LED
        ezTurnOffLED(2);

} // End of function: Init_and_Send_buffer_to_UART


/*********************************************************************

        Function:               Init_and_Receive_buffer_from_UART

*********************************************************************/

void Init_and_Receive_buffer_from_UART(
        ADI_DCB_HANDLE                  DCBManagerHandle_u,
        ADI_DMA_MANAGER_HANDLE  DMAManagerHandle_u,
        ADI_DEV_MANAGER_HANDLE  DeviceManagerHandle_u,
        ADI_DEV_1D_BUFFER               *pBuffer1D
)
{
        // table of configuration values for the UART
        ADI_DEV_CMD_VALUE_PAIR ConfigurationTable [] = {        // configuration table for the UART
driver
                {  ADI_DEV_CMD_SET_DATAFLOW_METHOD,     (void *)ADI_DEV_MODE_CHAINED    },
                {  ADI_UART_CMD_SET_DATA_BITS,          (void *)8
        },
                {  ADI_UART_CMD_ENABLE_PARITY,          (void *)FALSE
},
                {  ADI_UART_CMD_SET_STOP_BITS,          (void *)1
        },
                {  ADI_UART_CMD_SET_BAUD_RATE,          (void *)BAUD_RATE
},
                {  ADI_UART_CMD_SET_LINE_STATUS_EVENTS, (void *)TRUE
},
                {  ADI_DEV_CMD_END,                                                     NULL
                        },
        };

        // Handle to the UART driver
        ADI_DEV_DEVICE_HANDLE   DriverHandle_u;

        //32 sended_bytes_uart=0;

        // turn on 4. LED
        ezTurnOnLED(3);

        // callback after buffer processed enabled
    pBuffer1D->CallbackParameter = pBuffer1D; //NULL; //pBuffer1D;

    //
    uart_processed1=0;

        // open the UART driver for bidirectional data flow
        ezErrorCheck(adi_dev_Open(DeviceManagerHandle_u, &ADIUARTEntryPoint, 0, NULL,
&DriverHandle_u, ADI_DEV_DIRECTION_BIDIRECTIONAL, NULL, DCBManagerHandle_u, Callback));

        // configure the UART driver with the values from the configuration table
        ezErrorCheck(adi_dev_Control(DriverHandle_u, ADI_DEV_CMD_TABLE, ConfigurationTable));


        // receive the inbound buffer 1D
        ezErrorCheck(adi_dev_Read(DriverHandle_u, ADI_DEV_1D, (ADI_DEV_BUFFER *)pBuffer1D));

        // enable data flow
        ezErrorCheck(adi_dev_Control(DriverHandle_u, ADI_DEV_CMD_SET_DATAFLOW, (void *)TRUE));

                //ezErrorCheck(adi_dev_Control(DriverHandle_u,
ADI_UART_CMD_COMPLETE_OUTBOUND_BUFFER, NULL));
                //ezErrorCheck(adi_dev_Control(DriverHandle_u,
ADI_UART_CMD_GET_OUTBOUND_PROCESSED_ELEMENT_COUNT, &sended_bytes_uart));
                //printf(" ADI_UART_CMD_GET_OUTBOUND_PROCESSED_ELEMENT_COUNT = %d\n",
sended_bytes_uart);

        // wait till the buffer has been processed
        while (pBuffer1D->ProcessedFlag == FALSE) ;
        //while (uart_processed1==0) ;

        uart_processed1=0;

        // wait for sending last byte before close UART driver
        ezDelay(300);

        // close the UART driver
        ezErrorCheck(adi_dev_Close(DriverHandle_u));


        ezDelay(300); // delay for LED

        // turn off 4. LED
        ezTurnOffLED(3);

} // End of function: Init_and_Receive_buffer_from_UART


/*********************************************************************

        Function:               Send_frame_to_UART

*********************************************************************/

/*void Send_frame_to_UART(ADI_DEV_1D_BUFFER *pBuffer1D)

{
        // turn on last LED
        ezTurnOnLED(EZ_LAST_LED);

        // since we're doing output, don't generate any callbacks
        pBuffer1D->CallbackParameter = NULL;

        // send the outbound buffer 1D
        ezErrorCheck(adi_dev_Write(DriverHandle_uart, ADI_DEV_1D, (ADI_DEV_BUFFER *)pBuffer1D));


        // enable data flow
        ezErrorCheck(adi_dev_Control(DriverHandle_uart, ADI_DEV_CMD_SET_DATAFLOW, (void *)TRUE));

        //ezDelay(100); // wait for sending last byte
        // wait till the buffer has been processed
        //while (pBuffer1D->ProcessedFlag == FALSE) ; // not function in UART

        // Completes processing of the current outbound buffer
        //ezErrorCheck(adi_dev_Control(DriverHandle_uart, ADI_UART_CMD_COMPLETE_OUTBOUND_BUFFER,
(void *)TRUE));

        ezDelay(1000);

                // close the UART driver
                ezErrorCheck(adi_dev_Close(DriverHandle_uart));


        // turn off last LED
        ezTurnOffLED(EZ_LAST_LED);


}*/

/*********************************************************************

        Function:               Open_UART

*********************************************************************/
```

```
/*void Open_UART()
{
        // table of configuration values for the UART
        ADI_DEV_CMD_VALUE_PAIR ConfigurationTable [] = {        // configuration table for the UART
driver
                { ADI_DEV_CMD_SET_DATAFLOW_METHOD,      (void *)ADI_DEV_MODE_CHAINED    },
                { ADI_UART_CMD_SET_DATA_BITS,           (void *)8
        },
                { ADI_UART_CMD_ENABLE_PARITY,           (void *)FALSE
},
                { ADI_UART_CMD_SET_STOP_BITS,           (void *)1
        },
                { ADI_UART_CMD_SET_BAUD_RATE,           (void *)BAUD_RATE
},
                { ADI_UART_CMD_SET_LINE_STATUS_EVENTS,  (void *)TRUE
},
                { ADI_DEV_CMD_END,                                              NULL
                        },
        };

        // open the UART driver for bidirectional data flow
        ezErrorCheck(adi_dev_Open(DeviceManagerHandle_uart, &ADIUARTEntryPoint, 0, NULL,
&DriverHandle_uart, ADI_DEV_DIRECTION_BIDIRECTIONAL, NULL, DCBManagerHandle_uart, Callback_UART));

        // configure the UART driver with the values from the configuration table
        ezErrorCheck(adi_dev_Control(DriverHandle_uart, ADI_DEV_CMD_TABLE, ConfigurationTable));

}*/


/*********************************************************************

        Function:               compare_float

*********************************************************************/

int compare_double (const void *a, const void *b)
{
        double aval = *(double *)a;
        double bval = *(double *)b;
        if (aval < bval)
                return -1;
        else if (aval == bval)
                return 0;
        else
                return 1;
}


/*********************************************************************

        Function:               Face_detect_color_based_from_ycbcr_frame

*********************************************************************/

void Face_detect_color_based_from_ycbcr_frame(void)
{
        // turn on 9. LED
        ezTurnOnLED(8);

        f1 = _YCbCrtoRGB; // function: Convert YCbCr to RGB from .asm file: YCbCrtoRGB.asm

        u8 rgb1[3];
        u8 ycbcr1[3];
        unsigned int px, ln, pxf;
        unsigned int i1=0;
        unsigned int i2=0;
        unsigned int i3=0;
        unsigned int i4=0;
        unsigned int i5=0;
        unsigned int ix5=0;
        unsigned int iy5=0;
        u8 minY=255;
        u8 maxY=0;
        u8 maxCn=0;
        unsigned int Yavg=0; // average Y
        unsigned int count1=0;

        typedef struct
        {
                unsigned int x;
                unsigned int y;
        } Struct_xy;

        Struct_xy box_position[MAX_N_BOX]; // position of boxes with potencional faces - first
MAX_N_BOX boxes in the frame

        u8 cond1=0;



        //--- Conversion from 720x480 ycbcr frame to 180x120 rgb frame (:4):
                //Frame_ycbcr[1440 x 525] -> rgb_reduced_frame[3x180x120]:

    for (ln=(ADI_ITU656_NTSC_ILF1_START-1); ln<(ADI_ITU656_NTSC_ILF1_END); ln=ln+2) // lines, every
2nd odd line (4th line in frame)
```

```
    {
        for (px=1 ; px<(NTSC_LINE_WIDTH_ACTIVE) ; px=px+8) // columns
        {
                pxf = ln * (NTSC_LINE_WIDTH_ACTIVE) + px; // byte in frame

                if ( ((px+1)%4)==0 ) // every 4th pixel - Y
                {
                        ycbcr1[0]=Frame_ycbcr[pxf]; // Y
                        ycbcr1[1]=Frame_ycbcr[pxf-3]; // Cb
                        ycbcr1[2]=Frame_ycbcr[pxf-1]; // Cr
                }
                else  // every 2th pixel - Y
                {
                        ycbcr1[0]=Frame_ycbcr[pxf]; // Y
                        ycbcr1[1]=Frame_ycbcr[pxf-1]; // Cb
                        ycbcr1[2]=Frame_ycbcr[pxf+1]; // Cr
                }

                f1(ycbcr1, rgb1, 1); // convert YCbCr to RGB

                rgb_reduced_frame[i1+0]=rgb1[0]; // R
                rgb_reduced_frame[i1+1]=rgb1[1]; // G
                rgb_reduced_frame[i1+2]=rgb1[2]; // B

                i1=i1+3;

                //--- Finding min.Y:
                        if (ycbcr1[0]<minY)
                        {
                                minY=ycbcr1[0];
                        }
                //---

                //--- Finding max.Y:
                        if (ycbcr1[0]>maxY)
                        {
                                maxY=ycbcr1[0];
                        }
                //---

                Y_reduced_frame[i2]=ycbcr1[0]; // array of Y
                i2=i2+1;
        }

    }
    //---


    //--- Normalizing of Y frame and Counting average Y
        for (i2=0; i2<(X_R_FRAME*Y_R_FRAME); i2=i2+1)
        {
                //Y=255.0*(Y-minY)./(maxY-minY);
                Y_reduced_frame[i2]= ( ((Y_reduced_frame[i2]-minY)*255) / (maxY-minY) ); //
normalizing the Y to be 0..255
                Yavg = Yavg+Y_reduced_frame[i2];
        }

        Yavg = Yavg / (X_R_FRAME*Y_R_FRAME);
    //---

        //-R-// Now is Y_reduced_frame[] normalized (0-255)

    float T=1.0;

    if (Yavg<64)
    {
        T=1.4;
    }

    if (Yavg>192)
    {
        T=0.6;
    }

    //--- Compensation R and G, and computing Cr ( !!! Cr[] = byte_array_reduced[] !!!)
        i1=0;
        for (i2=0; i2<(3*X_R_FRAME*Y_R_FRAME); i2=i2+3)
        {
                if (T==1)
                {
                        // Cr = 0.5R - 0.419G - 0.081B
                        Cr[i1] = ( 127*rgb_reduced_frame[i2+0] - 107*rgb_reduced_frame[i2+1] -
21*rgb_reduced_frame[i2+2] ) / 255; // computing Cr (Cr = byte_array_reduced[])
                }
                else
                {
                        // R'=R^T; G'=G^T; B=B
                        rgb_reduced_frame[i2+0]=powf((float)rgb_reduced_frame[i2+0],T); // refill
rgb_reduced_frame[] with compensated R' G' B
                        rgb_reduced_frame[i2+1]=powf((float)rgb_reduced_frame[i2+1],T);
                        rgb_reduced_frame[i2+2]=rgb_reduced_frame[i2+2];

                        // Cr = 0.5R' - 0.419G' - 0.081B
                        Cr[i1] = ( 0.5*((float)rgb_reduced_frame[i2+0]) -
0.419*((float)rgb_reduced_frame[i2+1]) - 0.081*((float)rgb_reduced_frame[i2+2]) ); // counting Cr
(Cr = byte_array_reduced[])
                }
                i1=i1+1;
        }
    //---

        //-R-// Now is rgb_reduced_frame[R,G,B] compensated

    //--- Skin detection:
        for (i2=0; i2<(X_R_FRAME*Y_R_FRAME); i2=i2+1)
        {
                if ( (Cr[i2]>10) && (Cr[i2]<45) )
                {
                        Skin[i2]=SKIN; // skin pixel; !!! Skin[]=byte_array_reduced[] !!!
                }
                else
                {
                        Skin[i2]=NON_SKIN; // non skin pixel
                }
        }

    //---

        //-R-// Skin[] - frame of skin pixels

    //--- High frequency noisy removing - averaging in DIV_N x DIV_N area (5x5):
Skin[all]=(DIV_N*DIV_N)*Skin_noise_remove[all]

        for (i1=0, iy5=0 ; i1<(Y_R_FRAME); i1=i1+DIV_N, iy5=iy5+1) // in y
        {

                for (i2=0, ix5=0 ; i2<(X_R_FRAME); i2=i2+DIV_N, ix5=ix5+1) // in x
                {
                        count1=0;

                        for (i3=0; i3<(DIV_N); i3=i3+1) // in y (DIV_N area)
                        {
                                for (i4=0; i4<(DIV_N); i4=i4+1) // in x (DIV_N area)
                                {
                                        if ( Skin[i1*X_R_FRAME+i2+i3*X_R_FRAME+i4]==SKIN ) // if
skin pixel
                                        {
                                                count1=count1+1;
                                        }
                                             //povodne//
Skin[i1*X_R_FRAME+i2+i3*X_R_FRAME+i4]=NON_SKIN; // set to non skin pixel
                                }
                        }

                        //povodne:
                        /*if ( count1 > ( (DIV_N*DIV_N)/2 ) ) // if in 5x5 area is more skin pixel,
then fill all 5x5 area of skin pixels
                        {
                                for (i3=0; i3<(DIV_N); i3=i3+1) // in DIV_N area
                                {
                                        for (i4=0; i4<(DIV_N); i4=i4+1) // in DIV_N area
                                        {
                                                Skin[i1*X_R_FRAME+i2+i3*X_R_FRAME+i4]=0; // set to
skin pixel
                                        }
                                }
                        }*/


                        if ( count1 > ( (DIV_N*DIV_N)/2 ) ) // if in 5x5 area is more skin pixel,
then fill all 5x5 (one area) of skin pixels
                        {
                                Skin_noise_remove[iy5*X_R_REM_FRAME+ix5]=SKIN; // for testing //
Skin area
                                Skin_noise_remove_2D[iy5][ix5]=SKIN; // Skin area
                        }
                        else
                        {
                                Skin_noise_remove[iy5*X_R_REM_FRAME+ix5]=NON_SKIN; // for testing //
non skin area
                                Skin_noise_remove_2D[iy5][ix5]=NON_SKIN; // non skin area
                        }

                }

        }

    //---

        //-R-// Skin_noise_remove_2D[][] - frame of skin (area of pixels with the same skin) - after
noise removing

    //--- Finding blocks of potencional faces (Box: F_BOX_X x F_BOX_Y):

                        //povodne//conv2d_fr16(Skin_noise_remove_2D, Y_R_REM_FRAME, X_R_REM_FRAME,
array_conv, F_BOX_Y, F_BOX_X, out_conv_array); // convolution Skin_noise_remove_2D array with 7x7
array of SKIN pixel to find 7x7 block with skin pixels


                         // output is out_conv_array which contain number where
                        /*void conv2d_fr16(const fract16 *input_x,
                                        int rows_x,
                                        int columns_x,
                                const fract16 *input_y,
                                int rows_y,
                                int columns_y,
                                fract16 *output);*/

                maxCn=0;

                for (i1=0; i1<(MAX_N_BOX); i1=i1+1)
                {
                        box_position[i1].x = 0;
                        box_position[i1].y = 0;
                }

                // Scanning and counting skin pixels in boxes: F_BOX_X x F_BOX_Y:
                // Save position of first MAX_N_BOX (3) with most skin pixels:
                for (iy5=0; iy5<(Y_R_REM_FRAME - F_BOX_Y + 1); iy5=iy5+1)
                {
                        for (ix5=0; ix5<(X_R_REM_FRAME - F_BOX_X + 1); ix5=ix5+1)
                        {
                                count1=0;
                                for (i3=0; i3<(F_BOX_Y); i3=i3+1)
                                {
                                        for (i4=0; i4<(F_BOX_X); i4=i4+1)
                                        {
                                                if (Skin_noise_remove_2D[iy5+i3][ix5+i4]==SKIN)
                                                {
                                                        count1=count1+1;
                                                }
                                        }

                                }

                                if (count1>maxCn) // if new maximum of skin pixels in box
                                {
                                        maxCn=count1;

                                        for (i1=(MAX_N_BOX-1); i1>0; i1=i1-1) // fill positions of
boxes (position left-top of box)
                                        {
                                                box_position[i1].x = box_position[i1-1].x;
                                                box_position[i1].y = box_position[i1-1].y;
                                        }

                                        box_position[0].x = ix5; // box_position[0] -> position of
box with the most skin pixels
                                        box_position[0].y = iy5; // box_position[1] -> less pixels,
etc., till box_position[MAX_N_BOX]

                                }
                        }
                }

                // Mark detected faces to the Skin_noise_remove[] frame:
                ix5=box_position[0].x;
                iy5=box_position[0].y;

                Skin_noise_remove[iy5*X_R_REM_FRAME+ix5]=200; //mark of detected faces (half
intesity in grayscale)

                // Create face frame in grayscale from Y_reduced_frame[]:
                for (i1=0; i1<(F_BOX_Y*DIV_N); i1=i1+1)
                {
                        for (i2=0; i2<(F_BOX_X*DIV_N); i2=i2+1)
                        {
                                Face_one_Y_u8[i1*(F_BOX_X*DIV_N)+i2] = Y_reduced_frame[
(iy5*DIV_N+i1)*X_R_FRAME + ix5*DIV_N+i2 ];
                        }
                }

    //---

    //--- Normalizing of detected faces Face_one_Y_u8[]:

        u8 minYd=255;
                u8 maxYd=0;

        // Finding min.Y and max.Y
                for (i1=0; i1<(FACE_DET_SIZE); i1=i1+1)
                {

                        //--- Finding min.Y:
                if ( Face_one_Y_u8[i1]<minYd )
                {
                        minYd = Face_one_Y_u8[i1];
                }
                //---

                //--- Finding max.Y:
                if ( Face_one_Y_u8[i1] > maxYd )
                {
                        maxYd = Face_one_Y_u8[i1];
                }
                //---
                }

        // Normalize:
                for (i1=0; i1<(FACE_DET_SIZE); i1=i1+1)
                {
                        Face_one_Y_u8[i1] = ( ((Face_one_Y_u8[i1]-minYd)*255) / (maxYd-minYd) ); //
normalizing the Face1_Y[num_faces][i1] to be 0..255

                        Face1_Y[num_faces][i1] = (double)Face_one_Y_u8[i1];
                }

    //---

    // turn off 9. LED
        ezTurnOffLED(8);

        //-R-// box_position[i1].x
                //-R-// box_position[i1].y - position of blocks of potencional faces (position
left-top of box)



} // End of function: Face_detect_color_based_from_ycbcr_frame()
  // Output: double Face1_Y[][]: NUMBER_FACES x FACE_DET_SIZE
  //             u8 Face_one_Y_u8[]: FACE_DET_SIZE (current face)


/***********************************************************************

        Function:               Test_Eigen_solve_Matrix_multiply

***********************************************************************/

void Test_Eigen_solve_Matrix_multiply(void)
{
        #define N 3

    double A[N][N]={ {1,3,5}, {3,1,9}, {5,9,1} };
    double eigenvalues[N];
    double eigenvalues2[N][N];
    double eigenvectors[N][N];
    double vysl1[N][N];
    double vysl2[N][N];

    double X[2][3]={ {1,2,3}, {4,5,6} };
    double Y[3][2]={ {2,3}, {4,5}, {6,7} };
    double YV[3][1]={ {2}, {4}, {6} };

    double Ve[2];
    double Ve2[2][1];
    double Ve3[1][2];
    double Z[2][2];

    unsigned int a1=0;
    unsigned int a2=0;

    //test:
    double d_pok=(-12.12345);
    u8 u8_pok=72;


     //u8_pok = (u8) d_pok;
     d_pok = (double)u8_pok;


    // A*V = l*V
        Jacobi_Cyclic_Method(eigenvalues, (double*)eigenvectors, (double*)A, N);

        for (a1=0; a1<N; a1=a1+1)
        {
                for (a2=0; a2<N; a2=a2+1)
                {
                        eigenvalues2[a1][a2]=0;
                }
        }

        for (a1=0; a1<N; a1=a1+1)
        {
                eigenvalues2[a1][a1]=eigenvalues[a1];
        }

        printf( "eigenvalues[N], eigenvectors[N][N]: %f %f %f ;; %f %f %f ; %f %f %f ; %f %f %f\n",
eigenvalues[0], eigenvalues[1], eigenvalues[2], eigenvectors[0][0], eigenvectors[0][1],
eigenvectors[0][2], eigenvectors[1][0], eigenvectors[1][1], eigenvectors[1][2], eigenvectors[2][0],
eigenvectors[2][1], eigenvectors[2][2] );

        matmult(*A, N, N, *eigenvectors, N, *vysl1); // vysl1 = A * eigenvectors
        matmult(*eigenvalues2, N, N, *eigenvectors, N, *vysl2); // vysl2 = eigenvalues2 *
eigenvectors

        // A*V = l*V
        // ( A * eigenvectors = eigenvalues2 * eigenvectors )  =>  ( vysl1 == vysl1 )

        printf( "vysl1[N][N]: %f %f %f %f %f %f %f %f %f\n", vysl1[0][0], vysl1[0][1], vysl1[0][2],
vysl1[1][0], vysl1[1][1], vysl1[1][2], vysl1[2][0], vysl1[2][1], vysl1[2][2] );

        printf( "vysl2[N][N]: %f %f %f %f %f %f %f %f %f\n", vysl2[0][0], vysl2[0][1], vysl2[0][2],
vysl2[1][0], vysl2[1][1], vysl2[1][2], vysl2[2][0], vysl2[2][1], vysl2[2][2] );


/*      matmult(*X, 2, 3, *Y, 2, *Z);
```

```
        matmmlt(*X, 2, 3, *YV, 1, *Ve2);

        matmmlt(*X, 2, 3, *YV, 1, Ve);

        matmmlt(*X, 2, 3, *YV, 1, *Ve3);

        printf( "X: %f %f %f %f %f %f\n", X[0][0], X[0][1], X[0][2], X[1][0], X[1][1], X[1][2] );

        printf( "Z: %f %f %f %f\n", Z[0][0], Z[0][1], Z[1][0], Z[1][1] );
        printf( "Ve2[2][1]: %f %f\n", Ve2[0][0], Ve2[1][0] );
        printf( "Ve[2]: %f %f\n", Ve[0], Ve[1] );
        printf( "Ve3[1][2]: %f %f\n", Ve3[0][0], Ve3[0][1] );

*/

        printf( "d_pok, u8_pok: %f -> %u\n", d_pok, u8_pok);


} // End of function: Test_Eigen_solve_Matrix_multiply


/*********************************************************************

        Function:               Create_eigenface_database

*********************************************************************/

void Create_eigenface_database(void)
{
        // turn on 10. LED
        ezTurnOnLED(9);

        unsigned int pix1=0;
        unsigned int f1=0;
        unsigned int a1=0;
        unsigned int poml=0;
        double acc_double=0;
        unsigned int pom_p[NUMBER_FACES];


        // Face1_Y[][] ( NUMBER_FACES x FACE_DET_SIZE )

        //--- Mean face from detected faces:
                for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
                {
                        Mean_face[pix1]=0;

                        for (f1=0; f1<NUMBER_FACES; f1=f1+1)
                        {
                                Mean_face[pix1] = Face1_Y[f1][pix1] + Mean_face[pix1];
                        }

                        Mean_face[pix1] = ( Mean_face[pix1] / NUMBER_FACES );
                }
        //---

        //--- Subtract the mean face:
                for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
                {
                        for (f1=0; f1<NUMBER_FACES; f1=f1+1)
                        {
                                Face1_Y[f1][pix1] = Face1_Y[f1][pix1] - Mean_face[pix1];
                        }
                }
        //---

        //--- Create transposed matrix of Face1_Y[][]:
                for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
                {
                        for (f1=0; f1<NUMBER_FACES; f1=f1+1)
                        {
                                Face1_Y_transpose[pix1][f1] = Face1_Y[f1][pix1];
                        }
                }
        //---

        //--- Create Covariance matrix C' (modified):
                matmmlt(*Face1_Y, NUMBER_FACES, FACE_DET_SIZE, *Face1_Y_transpose, NUMBER_FACES,
*Covariance_matrix_m); // matrix * matrix multiplication: Covariance_matrix_m[][] = Face1_Y[][] *
Face1_Y_transpose[][] ( NUMBER_FACES x NUMBER_FACES )
        //---

        //--- Counting eigenvectors (modified, NUMBER_FACES x NUMBER_FACES) and eigenvalues
(NUMBER_FACES x 1) from Covariance matrix C' (Jacobi's cyclic method):
              // A*V = l*V
                // Jacobi_Cyclic_Method(eigenvalues, (double*)eigenvectors, (double*)A, N);
                // ( Covariance_matrix_m[][] * Eigenvectors_m[][] = Eigenvalues[] *
Eigenvectors_m[][] )

                Jacobi_Cyclic_Method(Eigenvalues, (double*)Eigenvectors_m,
(double*)Covariance_matrix_m, NUMBER_FACES); // Jacobi's cyclic method in "jacobi_cyclic_method.c"

        //---

        //--- Create true eigenvectors (NUMBER_FACES x FACE_DET_SIZE):
                //povodne-asi-chyba// matmmlt(*Face1_Y, NUMBER_FACES, FACE_DET_SIZE,
*Eigenvectors_m, NUMBER_FACES, *Eigenvectors); // matrix * matrix multiplication: Eigenvectors[][] =
Face1_Y[][] * Eigenvectors_m[][]
                matmult(*Eigenvectors_m, NUMBER_FACES, NUMBER_FACES, *Face1_Y, FACE_DET_SIZE,
*Eigenvectors); // matrix * matrix multiplication: Eigenvectors[][] = Eigenvectors_m[][] *
Face1_Y[][]
        //---

        //--- Sorting of eigenvalue (max -> min) and save original indexes:
                for (f1=0; f1<NUMBER_FACES; f1=f1+1)
                {
                        P_Eigenvalues[f1] = Eigenvalues[f1]; // Saving unsorted array of
Eigenvalues[]
                }

                qsort (Eigenvalues, NUMBER_FACES, sizeof(Eigenvalues[0]), compare_double); // sort
Eigenvalues[] in (min -> max) order

                for (f1=0; f1<NUMBER_FACES; f1=f1+1)
                {
                        for (a1=0; a1<NUMBER_FACES; a1=a1+1)
                        {
                                if ( Eigenvalues[f1] == P_Eigenvalues[a1] )
                                {
                                        pom_p[NUMBER_FACES-1-f1]=a1; // index of unsorted
Eigenvalues in sorted Eigenvalues
                                }

                        }
                }
        //---

        //--- Normalisation of Eigenvectors to has one magnitude:
                for (f1=0; f1<NUMBER_FACES; f1=f1+1)
                {
                        acc_double=0;

                        for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
                        {
                                acc_double = acc_double + ( Eigenvectors[f1][pix1] *
Eigenvectors[f1][pix1] );
                        }

                        acc_double = sqrt(acc_double);

                        for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
                        {
                                Eigenvectors[f1][pix1] = ( Eigenvectors[f1][pix1] / acc_double );
                        }
                }
        //---

        //--- Create transposed matrix of Eigenvectors[][] and sort by sorted values of
eigenvalues[]:
                for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
                {
                        for (f1=0; f1<NUMBER_FACES; f1=f1+1)
                        {
                                pom1 = pom_p[f1]; // conversion index for sorting Eigenvectors[][]
by sorted values of eigenvalues[]
                                Eigenvectors_transpose[pix1][f1] = Eigenvectors[pom1][pix1]; //
transponsing
                        }
                }
        //---

        //--- Creating Eigenface database - implement detected faces into the eigenspace:
                for (f1=0; f1<NUMBER_FACES; f1=f1+1)
                {
                        for (a1=0; a1<FACE_DET_SIZE; a1=a1+1)
                        {
                                One_face[0][a1] = Face1_Y[f1][a1];
                        }

                        matmult(*One_face, 1, FACE_DET_SIZE, *Eigenvectors_transpose, NUMBER_FACES,
*One_eigenface); // matrix * matrix multiplication: One_eigenface[] = Face1_Y[1][] *
Eigenvectors_transpose[][]

                        for (a1=0; a1<NUMBER_FACES; a1=a1+1)
                        {
                                Eigenface_database[f1][a1] = One_eigenface[0][a1];
                        }
                }
        //---

        // turn off 10. LED
        ezTurnOffLED(9);

} // End of function: Create_eigenface_database
  // Output: Eigenface_database[][]: NUMBER_FACES x NUMBER_FACES


/*********************************************************************

        Function:               Face_recognition

*********************************************************************/
```

```
void Face_recognition(void)
{
	// Input: Face1_Y[num_faces][] (current detected face)

	// turn on 11. LED
	ezTurnOnLED(10);

	unsigned int pix1=0;
	unsigned int f1=0;
	unsigned int a1=0;
	unsigned int pom1=0;
	double acc_double=0;


	//--- Subtract the mean face (current_face - Mean_face[]):
		for (pix1=0; pix1<FACE_DET_SIZE; pix1=pix1+1)
		{
			One_face[0][pix1] = Face1_Y[num_faces][pix1] - Mean_face[pix1];
		}
	//---

	//--- Creating Eigenface of unknown face - implement detected face into the eigenspace of
Eigenface database:

		matmmlt(*One_face, 1, FACE_DET_SIZE, *Eigenvectors_transpose, NUMBER_FACES,
*One_eigenface); // matrix * matrix multiplication: One_eigenface[] = Face1_Y[1][] *
Eigenvectors_transpose[][]

	//---

	//--- Classifying the unknown faces - Euclidean Distance (distance between unknown eigenface
and all eigenfaces from Eigenface database):
		for (f1=0; f1<NUMBER_FACES; f1=f1+1)
		{
			acc_double=0;

			for (a1=0; a1<NUMBER_FACES; a1=a1+1)
			{
				acc_double = acc_double + (
(One_eigenface[0][a1]-Eigenface_database[f1][a1]) *
(One_eigenface[0][a1]-Eigenface_database[f1][a1]) );
			}

			Distance_recognition[f1] = sqrt(acc_double); // Euclidean Distance between
unknown eigenface and f1-th eigenface from Eigenface database

			Distance_recognition_u8[f1] = (u8) ( Distance_recognition[f1] / 4 ); //
convert to be seen in "u8" range
		}

		for (f1=NUMBER_FACES; f1<MAX_FACES_IN_MEMORY; f1=f1+1)
		{
			Distance_recognition[f1]=DBL_MAX; // fill rest of array with max-distance
number
			Distance_recognition_u8[f1]=255;
		}
	//---

/*	//--- Classifying the unknown faces - Hamming Distance (distance between unknown eigenface
and all eigenfaces from Eigenface database):
		for (f1=0; f1<NUMBER_FACES; f1=f1+1)
		{
			acc_double=0;

			for (a1=0; a1<NUMBER_FACES; a1=a1+1)
			{
				acc_double = acc_double + ( abs( One_eigenface[0][a1] -
Eigenface_database[f1][a1] ) );
			}

			Distance_recognition[f1] = sqrt(acc_double); // Euclidean Distance between
unknown eigenface and f1-th eigenface from Eigenface database

			Distance_recognition_u8[f1] = (u8) ( Distance_recognition[f1] / 4 ); //
convert to be seen in "u8" range
		}

		for (f1=NUMBER_FACES; f1<MAX_FACES_IN_MEMORY; f1=f1+1)
		{
			Distance_recognition[f1]=DBL_MAX; // fill rest of array with max-distance
number
			Distance_recognition_u8[f1]=255;
		}
*/	//---

	//test//
	printf( " Distance_recognition[]: 01: %f , 02: %f , 03: %f , 04: %f , 05: %f , 06: %f , 07:
%f , 08: %f , 09: %f , 10: %f , 11: %f , 12: %f , 13: %f , 14: %f , 15: %f\n",
Distance_recognition[0], Distance_recognition[1], Distance_recognition[2], Distance_recognition[3],
Distance_recognition[4], Distance_recognition[5], Distance_recognition[6], Distance_recognition[7],
Distance_recognition[8], Distance_recognition[9], Distance_recognition[10],
Distance_recognition[11], Distance_recognition[12], Distance_recognition[13],
Distance_recognition[14] );

	// turn off 11. LED
	ezTurnOffLED(10);
```

```
} // End of function: Face_recognition
  // Output: Distance_recognition[]: 1 x NUMBER_FACES
  //         Distance_recognition_u8[]: 1 x NUMBER_FACES


/*********************************************************************

        Function:               main

        Description:    This function is the starting point of the example.
                                It first calls the EZ-Kit utility functions to
                                initialize the asynchronous and flash memories on
                                the board, then makes sure all LEDs are off.  The
                                Interrupt Manager is then initialized and the
                                exception and hardware error events are hooked.  If
                                deferred callbacks are enabled by the macro at the
                                top of the file, the Deferred Callback Manager is
                                then initialized with one service, running at IVG 14,
                                being created.  The DMA Manager and then the Device
                                Manager are then initialized.  The function then
                                populates a buffer that is used for both PPI input
                                and PPI output.  The buffer is configured to process
                                a standard NTSC 480i frame.  The function then enters
                                a loop, continually calling the capture function to
                                capture a frame of data and then a display function
                                to transmit the frame to a display device.

                                Should the last pushbutton switch ever be pressed, the loop
                                will exit and close down the Device Manager, DMA
                                Manager and, if it was enabled, the Deferred Callback
                                Manager.

*********************************************************************/


void main(void) {


        ADI_DEV_2D_BUFFER               Buffer2D;                       // buffer to be sent
to PPI Driver
        u32                             ResponseCount;                  // response counter
        u32     fcclk, fsclk, fvco;                     // frequencies
        int a1=0;


        // clear any pending hardware error (Si anomaly 04-00-0066, TAR 24983)
        *((u32 *)ILAT) = 0x20;

        // initialize the EZ-Kit
        ezInit(1);


        //Initialize all LEDs:
        for (a1=0; a1<EZ_NUM_LEDS; a1=a1+1)
        {
                ezInitLED(a1);
        }

        //Initialize all buttons:
        for (a1=0; a1<EZ_NUM_BUTTONS; a1=a1+1)
        {
                ezInitButton(a1);
        }

        // run at maximum speed and get current frequencies
        ezErrorCheck(adi_pwr_SetFreq(0, 0, ADI_PWR_DF_NONE));
        ezErrorCheck(adi_pwr_GetFreq((void*)&fcclk, (void*)&fsclk, (void*)&fvco));

        // turn off all LEDs
        ezTurnOffAllLEDs();

        // initialize the Interrupt Manager and hook the exception and hardware error interrupts
        ezErrorCheck(adi_int_Init(NULL, 0, &ResponseCount, NULL));
        ezErrorCheck(adi_int_CECHook(3, ExceptionHandler, NULL, FALSE));
        ezErrorCheck(adi_int_CECHook(5, HWErrorHandler, NULL, FALSE));

        // initialize the Deferred Callback Manager and setup a queue
  #if defined(USE_DEFERRED_CALLBACKS)
        ezErrorCheck(adi_dcb_Init(&DCBMgrData[0], ADI_DCB_QUEUE_SIZE, &ResponseCount, NULL));
        ezErrorCheck(adi_dcb_Open(14, &DCBMgrData[ADI_DCB_QUEUE_SIZE], (ADI_DCB_ENTRY_SIZE)*4,
&ResponseCount, &DCBManagerHandle));
  #else
        DCBManagerHandle = NULL;
  #endif



        // initialize the DMA Manager
        ezErrorCheck(adi_dma_Init(DMAMgrData, sizeof(DMAMgrData), &ResponseCount, &DMAManagerHandle,
NULL));

        // initialize the Device Manager
        ezErrorCheck(adi_dev_Init(DevMgrData, sizeof(DevMgrData), &ResponseCount,
&DeviceManagerHandle, NULL));


        // Populate the buffer that we'll use for the PPI input and output
```

```
                //Active frame only
                //Frame[ADI_ITU656_NTSC_LINE_WIDTH * ADI_ITU656_NTSC_HEIGHT]; // 1716 x 525
                //Frame_A[ADI_ITU656_NTSC_WIDTH * ADI_ITU656_NTSC_HEIGHT]; // 720 x 525
                //Frame_ycbcr[1440 * ADI_ITU656_NTSC_HEIGHT]; // 1440 x 525
        Buffer2D.Data = Frame_ycbcr; //Frame_A;
        Buffer2D.ElementWidth = 2;
        Buffer2D.XCount = (NTSC_LINE_WIDTH_ACTIVE / 2);
        Buffer2D.XModify = 2;
        Buffer2D.YCount = LINES_FROM_FRAME_NTSC;
        Buffer2D.YModify = 2;
        Buffer2D.CallbackParameter = NULL;
        Buffer2D.pNext = NULL;


    // Buffer for send via UART
        OutboundBuffer.Data = Face_one_Y_u8;    //rgb_reduced_frame; //Face_one_Y_u8; //Frame_ycbcr;
//Face_one_Y_u8; //Face1_Y[0]; //Face1_Y; //Skin; //Skin_noise_remove; //byte_array_reduced;
//Y_reduced_frame; //rgb_reduced_frame; //Frame_ycbcr; //OutboundData; //Frame_A;
        OutboundBuffer.ElementCount = (FACE_DET_SIZE);    //X_R_FRAME*Y_R_FRAME*3; //(FACE_DET_SIZE);
//756000; //(FACE_DET_SIZE); //(X_R_REM_FRAME*Y_R_REM_FRAME);//(X_R_FRAME * Y_R_FRAME);
//X_R_FRAME*Y_R_FRAME*3; //(1440 * ADI_ITU656_NTSC_HEIGHT); // 1440 x 525 = 756000
//(ADI_ITU656_NTSC_WIDTH * ADI_ITU656_NTSC_HEIGHT); // 720 x 525 = 378000
        OutboundBuffer.ElementWidth = 1; //sizeof(Face1_Y[0][0]); //1
        OutboundBuffer.CallbackParameter = &OutboundBuffer; //NULL; //&OutboundBuffer;
        OutboundBuffer.ProcessedFlag = FALSE;
        OutboundBuffer.pNext = NULL;

        // Buffer for send array via UART
        OutboundBuffer2.Data = Distance_recognition_u8;
        OutboundBuffer2.ElementCount = MAX_FACES_IN_MEMORY; //(NUMBER_FACES);
        OutboundBuffer2.ElementWidth = 1;
        OutboundBuffer2.CallbackParameter = &OutboundBuffer2;
        OutboundBuffer2.ProcessedFlag = FALSE;
        OutboundBuffer2.pNext = NULL;

        // Buffer for send array via UART
        OutboundBuffer3.Data = Array_u8;
        OutboundBuffer3.ElementCount = (FACE_DET_SIZE); //(NUMBER_FACES);
        OutboundBuffer3.ElementWidth = 1;
        OutboundBuffer3.CallbackParameter = &OutboundBuffer3;
        OutboundBuffer3.ProcessedFlag = FALSE;
        OutboundBuffer3.pNext = NULL;

        // Buffer for receive from UART
        InboundBuffer.Data = Face_one_Y_u8; //&InboundData;
        InboundBuffer.ElementCount = (FACE_DET_SIZE);
        InboundBuffer.ElementWidth = 1;
        InboundBuffer.CallbackParameter = &InboundBuffer;
        InboundBuffer.ProcessedFlag = FALSE;
        InboundBuffer.pNext = NULL;


// Start program:

        while (1) // Main cyclus
        {

                /// indicate num_faces on LEDs (number faces in database):

                if (num_faces<16)
                {
                        // turn on LED
                        ezTurnOnLED(num_faces);

                        // delay
                        ezDelay(50);

                        // turn off LED
                        ezTurnOffLED(num_faces);

                        // delay
                        ezDelay(50);
                }

                if ( (num_faces>=16) && (num_faces<31) )
                {
                        // turn on LED
                        ezTurnOnLED(0);
                        ezTurnOnLED(num_faces-15);

                        // delay
                        ezDelay(50);

                        // turn off LED
                        ezTurnOffLED(0);
                        ezTurnOffLED(num_faces-15);

                        // delay
                        ezDelay(50);
                }


                //// Capture frame from camera, face detect, send to UART:
                if (ezIsButtonPushed(BUTTON_SW6) == TRUE)
                {
                        // capture a frame of video
                        CaptureFrame(DCBManagerHandle, DMAManagerHandle, DeviceManagerHandle,
&Buffer2D);

                        num_faces=0; // clear pointer of detected face

                        // Face detect:
                        Face_detect_color_based_from_ycbcr_frame();

                        // Send face detected frame to the UART
                        Init_and_Send_buffer_to_UART(DCBManagerHandle, DMAManagerHandle,
DeviceManagerHandle, &OutboundBuffer);


                        // display the frame of video
                        //DisplayFrame(DCBManagerHandle, DMAManagerHandle, DeviceManagerHandle,
&Buffer2D);

                        // delay
                        ezDelay(300);

                                //Test_Eigen_solve_Matrix_multiply(); // Test
                }


                //// Capture frame from camera, face detect, save detected faces to memory -
database of detected faces (max. faces: MAX_FACES_IN_MEMORY):
/*              if (ezIsButtonPushed(BUTTON_SW7) == TRUE)
                {
                        if (num_faces < MAX_FACES_IN_MEMORY)
                        {
                                // capture a frame of video
                                CaptureFrame(DCBManagerHandle, DMAManagerHandle,
DeviceManagerHandle, &Buffer2D);

                                // Face detect:
                                Face_detect_color_based_from_ycbcr_frame();

                                num_faces=num_faces+1; // increase pointer of detected face to
create Eigenface database

                                // delay
                                ezDelay(300);
                        }

                }
*/

                /// Create Eigenface database from saved detected faces in memory
                if (ezIsButtonPushed(BUTTON_SW7) == TRUE)
                {
                        // Create Eigenface database:
                        Create_eigenface_database();

                                //TEST// Send array to the UART:
                        /*      for (ig1=0; ig1<FACE_DET_SIZE; ig1=ig1+1)
                                {
                                        Array_u8[ig1] = (u8) Mean_face[ig1]; //(u8)
(Face1_Y[0][ig1]); //(u8)Mean_face[ig1];
                                }
                                Init_and_Send_buffer_to_UART(DCBManagerHandle, DMAManagerHandle,
DeviceManagerHandle, &OutboundBuffer3);
                        */

                        // delay
                        ezDelay(300);
                }


/*              /// Face-recognition of unknown face (captured from camera and detected) - compare
eigenface of unknown face with Eigenface database
                if (ezIsButtonPushed(BUTTON_SW8) == TRUE)
                {
                        // capture a frame of video
                        CaptureFrame(DCBManagerHandle, DMAManagerHandle, DeviceManagerHandle,
&Buffer2D);

                        // Face detect  of unknown face:
                        Face_detect_color_based_from_ycbcr_frame();

                        // Face-recognition of unknown face:
                        Face_recognition();

                        // delay
                        ezDelay(300);

                        // Send Distance_recognition[] array to the UART:

                        // delay
                        ezDelay(300);
                }
*/


                /// TEST: Face-recognition of unknown face (received from UART - test database from
PC) - compare eigenface of unknown face with Eigenface database
                if (ezIsButtonPushed(BUTTON_SW8) == TRUE)
                {
                        // Receive frame from UART
                        Init_and_Receive_buffer_from_UART(DCBManagerHandle, DMAManagerHandle,
DeviceManagerHandle, &InboundBuffer);

                        // Copy detected face to face database on last place
                        for (ig1=0; ig1<FACE_DET_SIZE; ig1=ig1+1)
                        {
                                Face1_Y[num_faces][ig1] = (double)Face_one_Y_u8[ig1];
                        }

                        // Face-recognition of unknown face:
                        Face_recognition();

                        // delay
                        ezDelay(300);

                        // Send Distance_recognition_u8[] array to the UART:
                        Init_and_Send_buffer_to_UART(DCBManagerHandle, DMAManagerHandle,
DeviceManagerHandle, &OutboundBuffer2);

                        // delay
                        ezDelay(300);
                }


                /// TEST: Receive small frame (detected face) from UART and copy to face database,
(send back to UART for verify)
                if (ezIsButtonPushed(BUTTON_SW9) == TRUE)
                {
                        // Receive frame from UART
                        Init_and_Receive_buffer_from_UART(DCBManagerHandle, DMAManagerHandle,
DeviceManagerHandle, &InboundBuffer);

                        // Copy detected face to face database
                        for (ig1=0; ig1<FACE_DET_SIZE; ig1=ig1+1)
                        {
                                Face1_Y[num_faces][ig1] = (double)Face_one_Y_u8[ig1];
                        }

                        if (num_faces < MAX_FACES_IN_MEMORY )
                        {
                                num_faces=num_faces+1; // increase pointer of detected face to
create Eigenface database
                        }

                        // delay
                        ezDelay(300);

                        // Send frame to the UART
                        Init_and_Send_buffer_to_UART(DCBManagerHandle, DMAManagerHandle,
DeviceManagerHandle, &OutboundBuffer);


                        // delay
                        ezDelay(300);
                }

        // ENDLOOP
        } // End: Main cyclus

        // close down the device driver
        //ezErrorCheck(adi_dev_Close(DriverHandle));

        // close the Device Manager
        ezErrorCheck(adi_dev_Terminate(DeviceManagerHandle));

        // close down the DMA Manager
        ezErrorCheck(adi_dma_Terminate(DMAManagerHandle));

        // close down the Deferred Callback Manager
#if defined(USE_DEFERRED_CALLBACKS)
        ezErrorCheck(adi_dcb_Terminate());
#endif

// return
} // End of function: main
```

```
//----- L3 Memory (SDRAM): -----

#include <services/services.h>      // include the system services
#include "../coreA/adi_itu656.h"    // ITU656 utilities

#include "../Header1.h"                             // my Header


// Create an area in SDRAM that will the NTSC frame
//u8 Frame[ADI_ITU656_NTSC_LINE_WIDTH * ADI_ITU656_NTSC_HEIGHT]; // 1716 x 525

//only active field video frame
//u8 Frame_A[ADI_ITU656_NTSC_WIDTH * ADI_ITU656_NTSC_HEIGHT]; // 720 x 525

//active field YCbCr: | Cb Y Cr | Y | Cb Y Cr | Y | ....
u8 Frame_ycbcr[NTSC_LINE_WIDTH_ACTIVE * LINES_FROM_FRAME_NTSC]; // 1440 x 525

u8 Skin_noise_remove[X_R_REM_FRAME*Y_R_REM_FRAME];
double Face1_Y[MAX_FACES_IN_MEMORY][FACE_DET_SIZE];
u8 Face_one_Y_u8[FACE_DET_SIZE];
double One_face[1][FACE_DET_SIZE];
double Face1_Y_transpose[FACE_DET_SIZE][MAX_FACES_IN_MEMORY];
double Mean_face[FACE_DET_SIZE];
double Covariance_matrix_m[MAX_FACES_IN_MEMORY][MAX_FACES_IN_MEMORY]; // zmenit na
[MAX_FACES_IN_MEMORY][MAX_FACES_IN_MEMORY]; !!!
double Eigenvalues[MAX_FACES_IN_MEMORY];
double P_Eigenvalues[MAX_FACES_IN_MEMORY];
double Eigenvectors_m[MAX_FACES_IN_MEMORY][MAX_FACES_IN_MEMORY];
double Eigenvectors[MAX_FACES_IN_MEMORY][FACE_DET_SIZE];
double Eigenvectors_transpose[FACE_DET_SIZE][MAX_FACES_IN_MEMORY];
double Eigenface_database[MAX_FACES_IN_MEMORY][MAX_FACES_IN_MEMORY];
double One_eigenface[1][MAX_FACES_IN_MEMORY];
double Distance_recognition[MAX_FACES_IN_MEMORY];
u8 Distance_recognition_u8[MAX_FACES_IN_MEMORY];
u8 Array_u8[FACE_DET_SIZE];

        // double = 32 bit //

//presunute zo sml2:
u8 rgb_reduced_frame[X_R_FRAME*Y_R_FRAME*3]; // R G B ... 3x180x120 = 64800 , in L2 memory: R1 G1 B1
R2 G2 B2
u8 byte_array_reduced[X_R_FRAME*Y_R_FRAME]; // 180x120, in L2 memory
u8 Y_reduced_frame[X_R_FRAME*Y_R_FRAME]; // 180x120, in L2 memory
        //u8 Skin_noise_remove[X_R_REM_FRAME*Y_R_REM_FRAME]; // , in L2 memory
u8 Skin_noise_remove_2D[Y_R_REM_FRAME][X_R_REM_FRAME]; // 36x24, in L2 memory
        //fract16 out_conv_array[Y_R_REM_FRAME+A_CONV_Y-1][X_R_REM_FRAME+A_CONV_X-1]; // in L2
memory

//---
```

```
//----- Header file for project ( my #define ) -----//

//Define:

        //#define USE_DEFERRED_CALLBACKS  // using deferred callbacks, if not, using normal
callbacks

        #define BAUD_RATE       (57600)  //(460800)   //(57600) // baud rate at which the UART will
run

        #define NTSC_LINE_WIDTH_ACTIVE 1440 // Active NTSC (ADI_ITU656) frame line width
        #define NTSC_HEIGHT 525                         // NTSC (ADI_ITU656) frame height including
active & blank lines
        #define NTSC_RESOLUTION_X 720           // NTSC (ADI_ITU656) X resolution
        #define NTSC_RESOLUTION_Y 480           // NTSC (ADI_ITU656) Active lines in a frame (Field1
+ Field2) - Y resolution
        #define NTSC_FIELD_LINES 240            // NTSC (ADI_ITU656) Active Field lines

        #define LINES_FROM_FRAME_NTSC ( NTSC_HEIGHT ) // number of lines to process from frame

        #define X_R_FRAME (180) // resolution x of redeced frame - pixels
        #define Y_R_FRAME (120) // resolution y of redeced frame - pixels

        #define DIV_N (5)       // dimension (x,y) for noise removing (area of pixels from skin
frame to noise remove)

        #define X_R_REM_FRAME ( X_R_FRAME / DIV_N ) //36// resolution x of redeced frame after noise
removing - pixels
        #define Y_R_REM_FRAME ( Y_R_FRAME / DIV_N ) //24// resolution y of redeced frame after noise
removing - pixels


        #define Skin byte_array_reduced
        #define Cr byte_array_reduced

        #define SKIN 255 // skin pixel    //0
        #define NON_SKIN 0 // non skin pixel    //255

        #define F_BOX_Y 7 // potencional face box Y-dimension
        #define F_BOX_X 7 // potencional face box X-dimension

        #define MAX_N_BOX 3 // maximum number of boxes (potencional faces) in the frame

        #define MAX_FACES_IN_MEMORY 20 // maximum number faces saved in memory
        //#define NUMBER_FACES (num_faces+1) // number of detected and processing faces saved in
memory
        #define NUMBER_FACES (num_faces) // number of detected and processing faces saved in memory


        #define FACE_DET_SIZE (F_BOX_Y*DIV_N * F_BOX_X*DIV_N) // 35*35

        ////

        #define BUTTON_SW6 0    // EZ_FIRST_BUTTON
        #define BUTTON_SW7 1
        #define BUTTON_SW8 2
        #define BUTTON_SW9 3    // EZ_LAST_BUTTON


//---
```