25 mars 2009

SP5 : Portage

Rapport -

Portage des algorithmes de reconnaissance faciale

Pascal PELLET, Philippe JOURDAIN

e2v semiconductors
Advanced Imaging Division
Tél : +33 (0) 476 583 000

Table of Contents

Introduction

Dans le cadres des sous projets SP5 et SP6, e2v semiconductors a porté les algorithmes développés durant le sous projet SP2 sur une plateforme représentative d'un système de téléphonie mobile ou d'un PDA.

La plate-forme choisie est présentée dans le chapitre 1. Le portage en langage C des algorithmes développés initialement sous Matlab a été réalisé selon la spécification de besoin donnée en annexe. Le chapitre 3 décrit succinctement la méthodologie de validation, et le chapitre 4 présente les premières conclusions sur ce portage.

1. Présentation de la plate-forme

La plate-forme est constituée d'un processeur ATMEL AT91SAM9263 basé sur un cœur 32-bit ARM9 cadencé à 200 MHz. Les applications visées par ce processeur sont les systèmes autonomes demandant de la puissance de calcul et autonomie. Il possède notamment pour notre application :

- une interface capteur d'images CMOS (vidéo) (contrôleur ISI)
- Une interface d'affichage (LCD)
- une interface audio.

Coté software, le système opérationnel Linux a été porté sur la plate-forme.

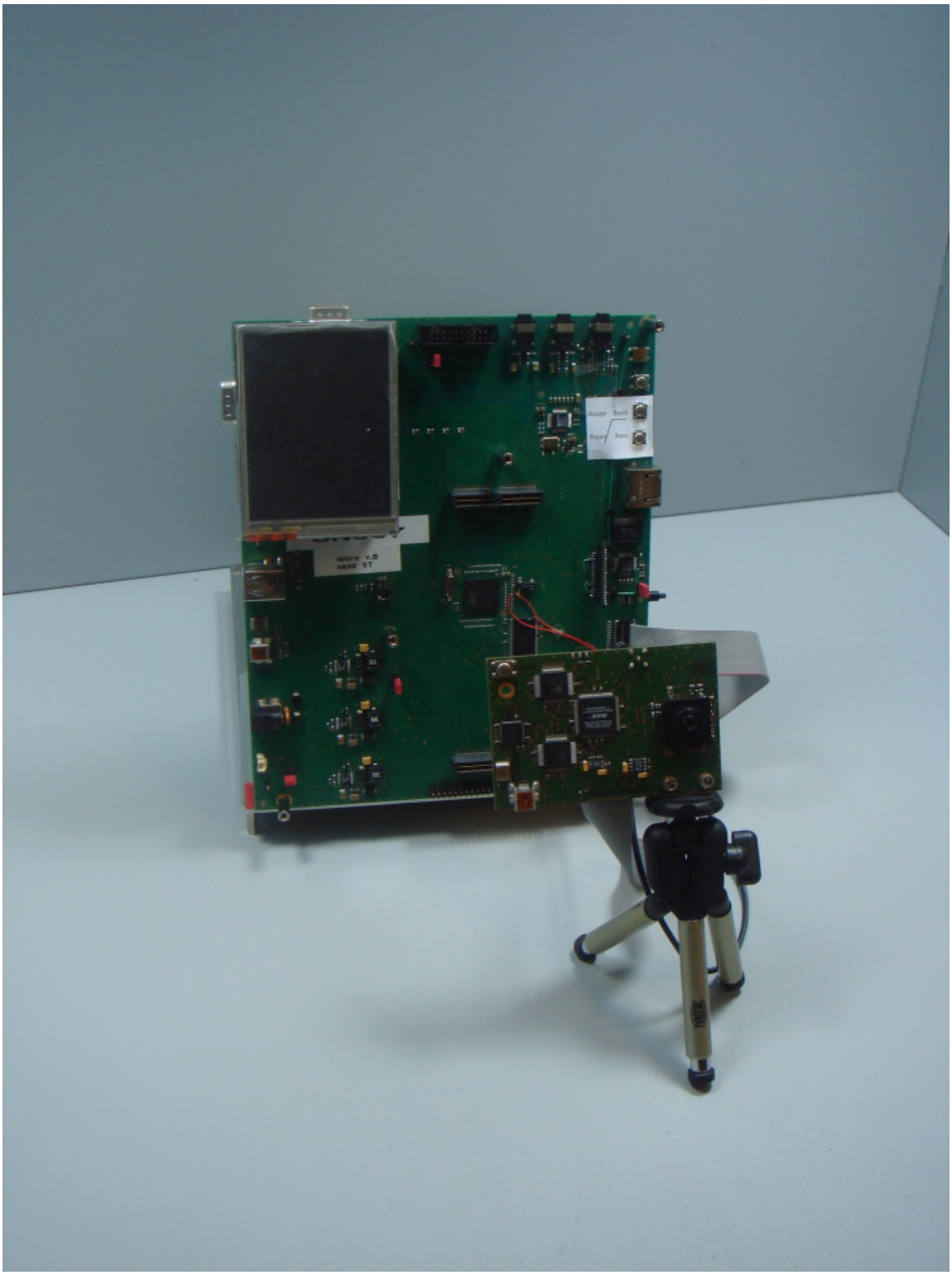
Ce processeur propose pour chacune des interfaces, un DMA (Direct Memory Access) qui permet un transfert des données soit du contrôleur ISI vers de la mémoire, soit de la mémoire vers le LCD. Ceci laisse tout le temps au processeur de traiter les images.

La vidéo est fournie par un capteur développé par e2v (3Megapixels).

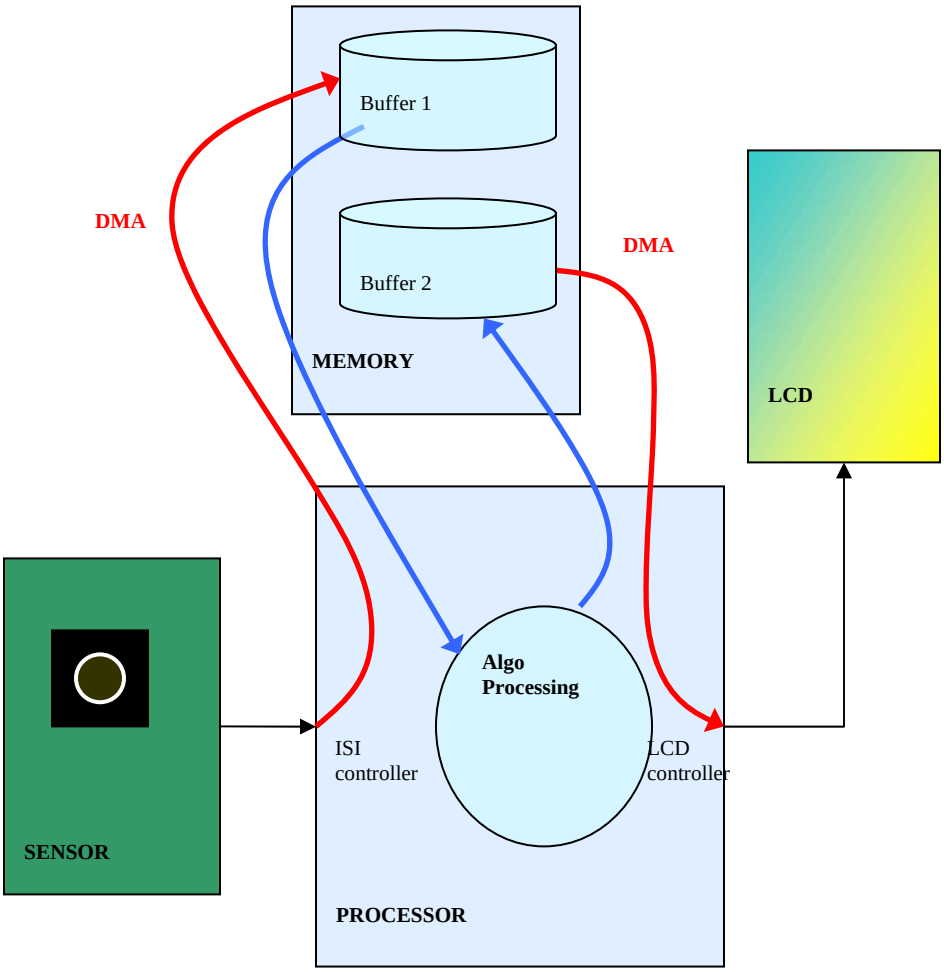
La configuration est la suivante :

- Taille de l'image en sortie : 320*240 (configuration des fonctions de ROI, « binning » et « subsampling » embarqués sur le capteur pour réduire le format de l'image)
- 30 fr/s

La photo ci-dessous montre la plate-forme développée ainsi que l’imageur



Le Diagramme ci-dessous représente l'écoulement du flot de données utilisé pour notre application :



2. Portage des algorithmes

Dans le cadre du SP2, une application a été développée sous Mathlab mettant en œuvre les algorithmes permettant la localisation des yeux, la création d'une image normalisée, l'enrôlement et l'identification d'une personne.

A partir de là, ces algorithmes ont été portés de Mathlab en langage C et optimisés pour pouvoir être exécutés sur la plate-forme.

La contrainte principale était de pouvoir tenir une fréquence image de 15 images par seconde.

Le flux vidéo est directement renvoyé sur le LCD avec seulement une correction GAMMA pour augmenter le contraste. Afin d'augmenter la performance, le gamma est fixe et donc on applique seulement une LUT (Look Up Table) à tous les pixels de l'image.

Ceci peut-se faire pendant la recopie du buffer 1 vers le Buffer 2 sans consommer de la ressource supplémentaire

Le seul calcul effectué en temps réel est la détection des yeux. Celle-ci est effectuée sur une zone prédéfinie de l'image (ROI) autour des yeux (appelée zone de détection).

Une fois les yeux sur le visage détectés, les autres fonctions (normalisation, enrôlement, reconnaissance) sont effectuées en tache de fond.

A la fin du codage, la fréquence image était inférieure à ce qui était souhaité.

Elle a été augmentée en modifiant une option de compilation (-O3)

L'empreinte mémoire pour cette application est :

2 buffers de la taille de l'image (320*240 octets) (correspondant au buffer1 et 2 du schéma ci-dessus

2 buffers pour le calcul de la zone de détection (zone de détection de l'image N et N-1)

Dans les fonctions de reconnaissance et d'enrôlement, des buffers sont temporairement alloués, de la taille de l'image normalisée.

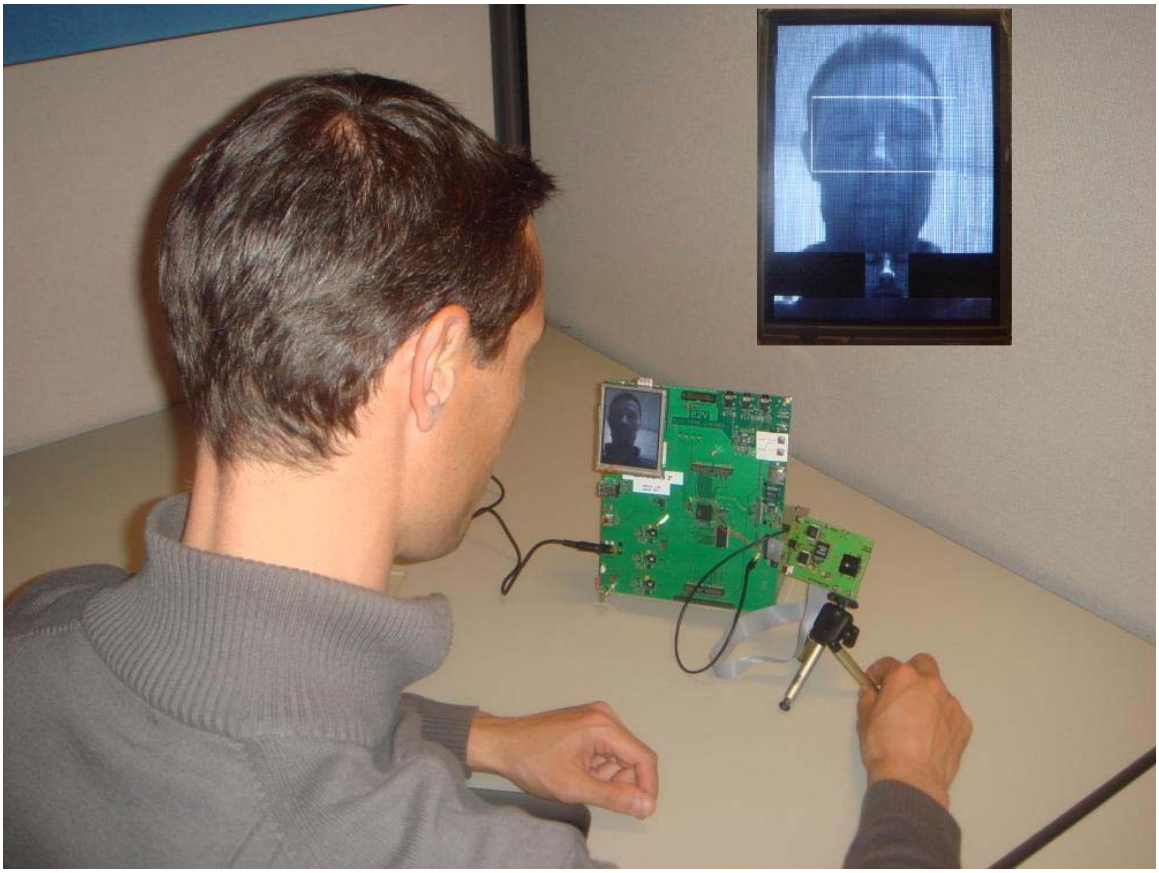
La spécification du portage des algorithmes se trouve en annexe.

3. Validation

Pour la validation du portage des algorithmes, chaque fonction portée a été validée unitairement en fournissant un « pattern » connu en entrée de la fonction et en comparant le résultat obtenu au résultat fourni par la fonction Matlab équivalente.

Les deux applications d'enrôlement et de reconnaissance sont fonctionnelles et ont déà fait l'objet d'une démonstration au sein de la société, plusieurs personnes ont ainsi pu s'enrôler et être reconnues avec succès.

Photo de la démonstration :



4. Conclusion

Les résultats obtenus démontrent la faisabilité d'une application de reconnaissance faciale sur un système de type téléphonie mobile. Les performances de la solution actuelle n'ont pas été évaluées sur un grand nombre de personnes, mais on peut déjà dégager deux grandes voies d'améliorations à explorer, notamment sur la détection du clignement des yeux.

Si on améliore la précision de la détection des yeux, les visages seront mieux normalisés en taille et en position, ainsi la reconnaissance sera d'autant plus efficace.

De même, le dispositif actuel n'inclut pas d'algorithme d'auto-exposition, un tel algorithme permettrait d'obtenir une exposition constante de la zone de détection et faciliterait grandement la normalisation en luminosité du visage extrait.

CONFIDENTIEL SOCIETE

ANNEXE (Confidential chapter)

Introduction

This document specifies all image processing functions, which have to be implemented in the BioBiMo demonstration application developed by the application design team. This demonstration platform is based on the AT91SAM9263 processor from ATMEL (ARM926-JS + Linux OS).

All processing functions have already been developed on Matlab version 7 release 14, for a complete PC demonstration. They have been simplified and adapted for the platform demonstration. The last step is to convert them into C language (ANSI standard). Some functions must be optimized to be as fast as possible due to “real time” needs of the application.

Chapter 3 gives a brief summary of the objective of BioBiMo application. Chapter 4 specifies the expected application. Chapter 5 specifies the image processing function expected in C language. For each function, a paragraph gives the matlab version of the function. When some adjustment / simplification can be done for the demonstration platform, they are directly indicated in this paragraph.

BioBiMo application presentation

BioBiMo application must demonstrate that facial recognition can be embedded on a “small” processor like the ARM926, which is representative of the range of processors available on mobile phone or PDA products. The objective is to verify that the person, which tries to use the device, is or is not the owner of the product. Access is denied if the verification fails.

Facial recognition was chosen among the other biometric techniques because the great majority of devices targeted by the application already integrate an image sensor, as well as video acquisition and image capture functions. Furthermore, people are more familiarized in the fact of taking a photo of itself than using others biometric technologies like iris or fingerprint recognition.

Facial recognition application needs two main applications:

- Enrolment application allows the user to create his identifier, often called template. This template must be stored in the device.

- Recognition application allows the user to be identified by the device. The device calculates the template of the person and compares it to the stored one. The access to the device is accepted or refused according to the difference between the new template and the stored one.

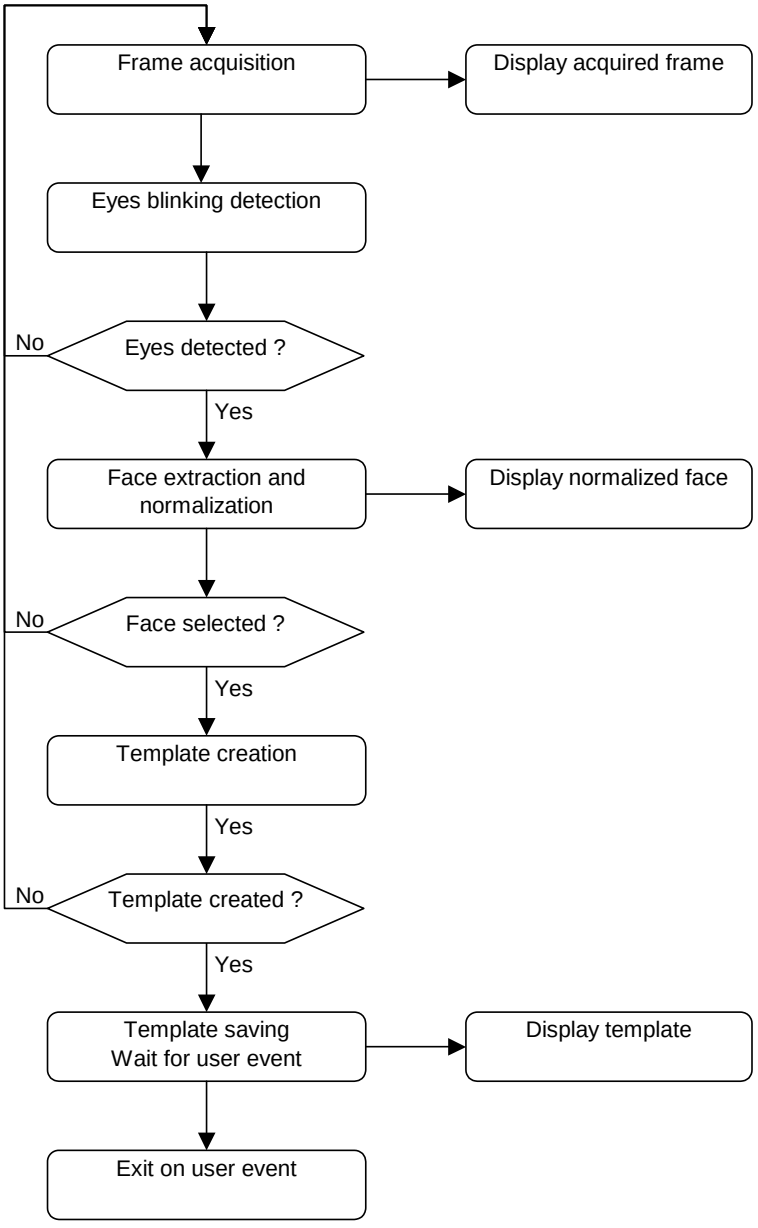
Facial recognition principle in BioBiMo application

The most difficult step in a facial recognition application is to detect correctly the face of the person. The e2v solution to this critical step consists into detecting eyes blinking in the video flow (patent FR2875322). The user must wink both eyes to be detected. When eyes positions are detected, it is easy to extract all the face of the person. Furthermore, as the user must do a special action to be detected, this solution is a good protection against impostor using e.g. photography of the owner.

Biobimo application specification

Enrolment specification

This flow chart describes the enrolment application principle:



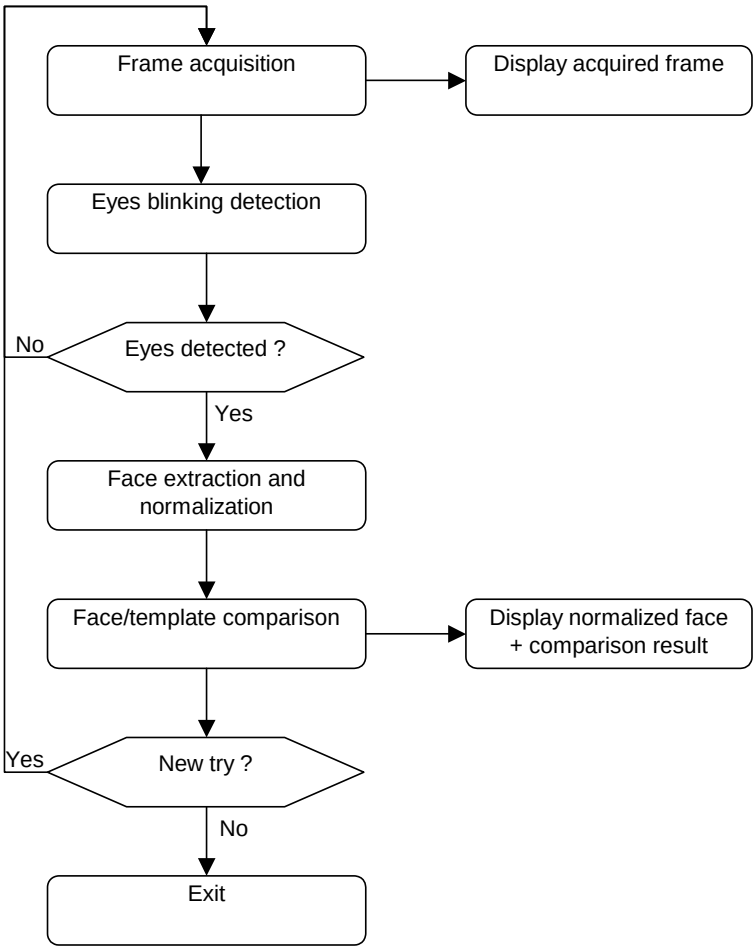
Enrolment application includes the following steps:

1. Video flow analysis to detect eyes blinking

2. Face extraction and normalization based on distance between eyes
3. Normalized face selection
4. Repetition of step 1 to 3 to accumulate a number of normalized faces. The template is calculated as the mean of the selected normalized faces.
5. Template saving

Recognition specification

This flow chart describes the recognition application principle:



Recognition application includes the following steps:

1. Video flow analysis to detect eyes blinking
2. Face extraction and normalization based on distance between eyes
3. Face comparison with stored template

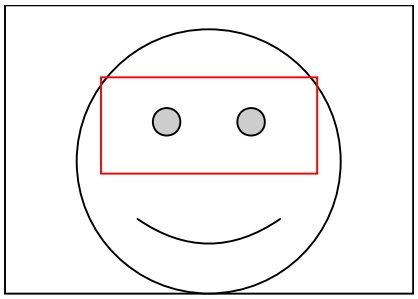
Steps 1 and 2 are identical to the enrolment application.

Display and frame acquisition management

During eyes blinking detection

During eyes blinking detection stage, the video screen of the platform must display the video flux. A red rectangle must be drawn in the image to help the user to position its eyes with regard to the camera and to limit the eyes blinking detection only on this area. This area is called further detection zone.

Rectangle positions and sizes must be configurable. Ideally, this rectangle must include only the high of the face as indicated below, because what happens in the background can perturb eyes blinking detection.



After a blinking detection

If a blinking is detected, the frame acquisition must be stopped.

For the enrolment application, the normalized face must be displayed and the user must push a button to accept it or not for its template creation. When its choice is made:

- If the number of selected normalized frame is not enough, the frame acquisition starts again.
- If the number of selected normalized frame is enough, the template is created and displayed.

The user must push a button to finish the application.

For the recognition application, the normalized face, plus the comparison result must be displayed.

The user must push a button to start a new recognition sequence or to finish the application.

Exit function

For both applications, the user must have the possibility to exit at any time.

Configuration parameters

Some parameters need to be configurable. These parameters must be defined in a configuration file.

Frame rate

A minimum frame rate of 10 fr/s (frame per second) is required to be able to detect eyes blinking. The targeted frame rate for this application is 15 fr/s.

Frame rate must be configurable.

Video format

QQVGA (120 lines x 160 columns) format is enough to perform facial recognition. But the video resolution depends on the sensor available for the demonstration platform.

A CIF (288 x 352) sensor is considered. This sensor includes sub-sampling function and ROI (Region of Interest) capabilities. A (240 x 320) ROI combined with a sub-sampling per 2 allows obtaining a QQVGA format.
ROI and sub-sampling function must be configurable.

Color and pixel resolution

Color is not requested for this application and 8 bits resolution is enough. The CIF sensor is a RGB color sensor, but it can be configured to obtain a black and white video data flow

Eyes blinking detection

Eyes blinking detection needs following configuration parameters:

- Minimum eyes space Maximum eyes space Detection zone configuration
  - Width must be equal the maximum eyes space
  - Height must be configurable (in general set to Width/2)
  - First column must be calculated to center the detection zone in the width of the configured ROI.
  - First row must be configurable.
- Minimum pixel number per line for a valid strip
- Minimum pixel number per column for a valid blob
- Minimum line number per strip for a valid strip
- Minimum pixel number per blob for a valid blob
- Threshold coefficient

Face normalization

The normalization in contrast uses a contrast factor. This factor must be configurable

(default value = 20).

Template and identifier

The template is a square area:

- Width = Height = 3/2 * minimum eyes space.

The number of normalized face zone required to make a template must be configurable. The input value defines a power of 2 to simplify the template creation step (FaceNb = $2^{configured\_value}$).

Template has to be saved in 8 bits black and white BMP format.

Both applications (enrolment and recognition) need an identifier. This identifier is used by the enrolment application to create the name of the template file and by the recognition application to open the template file. A four-digit number is enough.

Example of template name: TEMPLATE_0001.bmp

This identifier has to be configurable.

Face recognition

Face recognition uses a threshold to decide if a person is accepted or refused, this threshold must be configurable.

Memory needs

Enrolment application

Frame buffer requested:

1. One buffer for the frame acquisition (size depends on sensor configuration)
2. Two buffer of the same size for stored the previous and the current detection zone, buffer size in octets = detection zone width x detection zone height.
3. One buffer for the normalized face zone, buffer size in octets = (3/2 * minimum eyes space)²

4. One buffer for the template creation, buffer size in octets = 2 * (3/2 * minimum eyes space)²

Recognition application

Only the last buffer differs from the enrolment application, because the template read and stored in memory used only one octets per pixel instead of 2 for the template creation.

"Real time" needs

All function must be optimized and execution time must be as shorter as possible, but only the eyes blinking detection function must be executed on all video frames (15 fr/s). A maximum waiting time of 1 second to obtain the display of the normalized face will be satisfactory.

image Processing functions

This chapter specifies all image processing functions, expected for the C language version.

Detection zone extraction

Objective and principle

The GetDetectionZone function must extract the detection zone of an acquired frame

according to detection zone configuration parameters.

C function specification

Prototype

```
char GetDetectionZone(      unsigned char       *pucFrame,
                            unsigned char       *pucDetectZone,
                           t_sDetectZone        *psAlgoParam)
```

The function must returns:

- -1 if an error has been detected
- 0 if the detection zone has been extracted

Structure definition

Structure t_sDetectZone definition:

Field	Type	Comments
usFrameHeight	unsigned short	Height of the acquired frame (row number)
usFrameWidth	unsigned short	Width of the acquired frame (column number)
usDetectZoneX0	unsigned short	First column of the detection zone
usDetectZoneY0	unsigned short	First row of the detection zone
usDetectZoneHeight	unsigned short	Height of the detection zone
usDetectZoneWidth	unsigned short	Width of the detection zone

Input

Name	Type	Comments
pucFrame	Pointer on unsigned char	Pointer on a frame buffer containing a frame acquired from the sensor.
pucDetectZone	Pointer on unsigned char	Pointer on a frame buffer allocated to contain a detection zone
psAlgoParam	Pointer on structure t_sDetectZone	Structure containing all detection zone extraction parameters

Output

Name	Type	Comments
pucDetectZone	Pointer on unsigned char	The frame buffer is updated with the configured detection zone.

Matlab sources

In matlab, ROI extraction is done in a single instruction:

```
detect_zone = frame( y0:(y0+height-1), x0:(x0+width-1))
```

Eyes blinking detection

Objective and principle

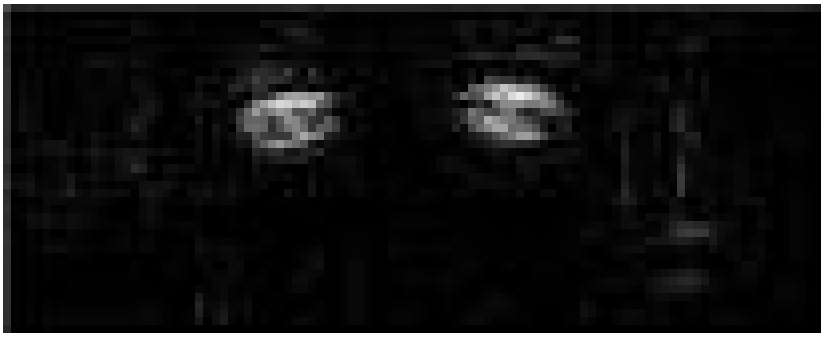
The DoEyesBlinkingDetection must analyze each detection zone and returns the position of both eyes, plus the space between eyes, if a blinking has been detected.

Blinking detection is based on a motion detection of eyelid. This detection is done in 2 main steps:

- Motion detection
- Eyes detection

Motion detection

Motion in a video can be detected by analyzing what is different in the current frame from the previous frame. Ideally, if the person doesn't move when it winks, there is noticeable different value between previous and current frame only around eyes position due to eyelid motion. The next figure shows the absolute value of the difference between each pixel of the previous frame and pixel of the current frame in the detection zone:

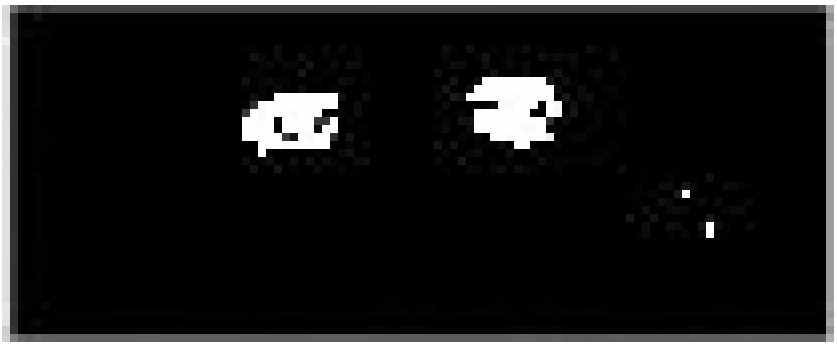


Motion picture

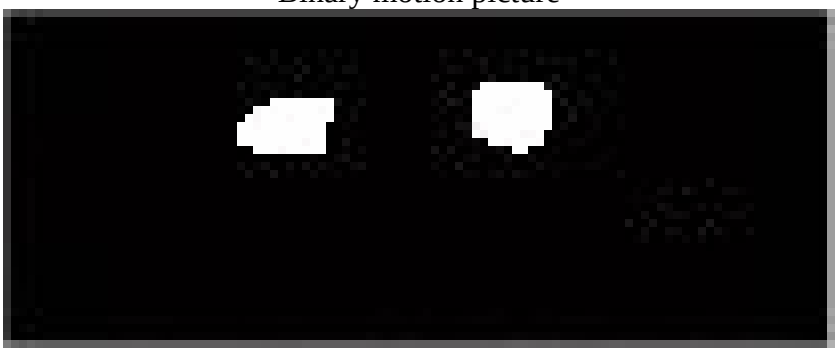
This picture, called motion picture, shows clearly two bright clusters due to eyelid motion. Some pre-processing are then done to isolate these clusters from noise:

1. Calculate a motion threshold from the mean of motion picture and a configurable coefficient.

2. Get a binary motion picture, by applying this threshold (see binary motion picture figure).
3. Do a 3x3 median filter on the binary motion picture to eliminate residual noise.



Binary motion picture



Binary motion picture after median filter

In the ideal case, it remains only 2 clusters called blob in the next paragraph. But if the person moves his head, or its mouth, or if something moves in the background, some parasitic blob can stay in the picture.

Eyes detection

The eyes detection step analyses the filtered binary motion picture to detect only 2 blobs.

The algorithm consists in finding first a valid strip:

- A valid strip must have a minimum number of successive valid lines (configurable).
- A valid line must have a minimum number of white pixels per line (configurable).
- For each column of the valid strip, the algorithm counts the number of white pixel per column.
- The first line of the valid strip is memorized.

When a valid strip is found, the algorithm detects the valid blobs included in this strip:

- A valid blob must have a minimum number of white pixels per column (configurable).
- A valid blob must have a minimum number of white pixels per blob (defined according to a configurable percent of the detection zone size).
- If more than 2 blobs are detected, the analysis jumps directly to the next line.
- The coordinates of first two blobs are memorized.

An eyes blinking is finally detected if:

- The detection zone contains only one valid strip with only 2 valid blobs.
- The distance between blobs must respect a configurable range.

C function specification

Prototype

```
char DoEyesBlinkingDetection(    unsigned char      *pucCurrentZone,
                                 unsigned char      *pucPreviousZone,
                                 t_sEyeBlkConf*psAlgoParams
                                 t_sEyeBlkRes       *psResults)
```

The function must returns:

- -1 if an error has been detected
- 1 if an eyes blinking has been detected
- 0 if no eyes blinking has been detected

Structure definition

Structure t_sEyeBlkConf definition:

Field	Type	Comments
usDetectZoneHeight	unsigned short	Height of detection zone (row number)
usDetectZoneWidth	unsigned short	Width of detection zone (column number)
usMinPixNbPerLine	unsigned short	Minimum pixel number per line for a valid strip
usMinPixNbPerColumn	unsigned short	Minimum pixel number per column for a valid blob
usMinLineNbPerStrip	unsigned short	Minimum line number per strip for a valid strip
usMinPixNbPerBlob	unsigned short	Minimum pixel number per blob for a valid blob
usMinEyesSpace	unsigned short	Minimum valid space between eyes
usMaxEyesSpace	unsigned short	Maximum valid space between eyes
ucThresholdCoef	unsigned char	Threshold coefficient

Structure t_sEyeBlkRes definition:

Field	Type	Comments
ausX[2]	area of 2 unsigned short	X coordinates for both eyes (column index)
ausY[2]	area of 2 unsigned short	Y coordinates for both eyes (row index)
usEyesSpace	unsigned short	Space between eyes

Input

Name	Type	Comments
pucCurrentZone	Pointer on unsigned char	Pointer on a frame buffer containing only the detection zone of the current frame
pucPreviousZone	Pointer on unsigned char	Pointer on a frame buffer containing only the detection zone of the previous frame
psAlgoParams	Pointer on structure t_sEyeBlkConf	Structure containing all eyes blinking detection parameters

Output

Name	Type	Comments
psResults	Pointer on structure t_sEyeBlkRes	Structure containing all eyes blinking detection results

Matlab sources

```
function [flag_ok, blob_data] = DoEyesBlinkingDetection( eyes_z1, eyes_z2, param)

% Objective :   Get eyes positions after an eyes blinking detection
%
% Inputs :
%   eyes_z1     image of size (DETECT_ZONE_HEIGHT,DETECT_ZONE_WIDTH) (current detection
zone)
%   eyes_z2     image of size (DETECT_ZONE_HEIGHT,DETECT_ZONE_WIDTH) (previous detection
zone)
%   param       structure of algorithm parameters
%
% Outputs :
%   flag_ok        Flag =  1 --> eyes detected,
%                          0 --> no eyes detected
%   blob_data      Structure with eyes coordinates + space between eyes
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% COMMENT FOR C VERSION :
% Warning in matlab, area index begin to 1 and not to 0 !!!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%

% Get algorithm parameters
DETECT_ZONE_HEIGHT      = param.DETECT_ZONE_HEIGHT;
DETECT_ZONE_WIDTH       = param.DETECT_ZONE_WIDTH;
MIN_PIX_NB_PER_LINE     = param.MIN_PIX_NB_PER_LINE;
MIN_PIX_NB_PER_COLUMN   = param.MIN_PIX_NB_PER_COLUMN;
MIN_LINE_PER_STRIP      = param.MIN_LINE_PER_STRIP;
MIN_PIX_NB_PER_BLOB     = param.MIN_PIX_NB_PER_BLOB;
MAX_EYES_SPACE          = param.MAX_EYES_SPACE;
MIN_EYES_SPACE          = param.MIN_EYES_SPACE;
THRESHOLD_COEF          = param.THRESHOLD_COEF;

% Initialise output values
blob_data   = [ ];
```

```
flag_ok     = 0;

% Do pre-processing
% -----------------

% Get frame difference
im_diff = uint8( abs(single(eyes_z1) - single(eyes_z2)) );

% Do binarisation
bin_th = THRESHOLD_COEF*mean2( im_diff );   % Get auto-adaptative threshold
im_diff( im_diff < bin_th )     = 0;
im_diff( im_diff >= bin_th )    = 1;

% Apply median filter to suppress isolated pixels
im_diff = medfilt2( im_diff, [3 3]);

% COMMENT FOR C VERSION :
% The mean can be calculated during the "Get frame difference" step
% The median filter on a binary picture consists in counting the white
% pixel number on a 3x3 kernel
% if this number is >= 5 the filter ouput is 1, else the filter output is 0
% A ring of 1 pixel cannot be processed



% Do eyes detection
% -----------------
col_sum = zeros(1,DETECT_ZONE_WIDTH);   % Initialise vertical accumulation vector
blob_nb = 0;                            % Initialise blob counter
row_nb  = 0;                            % Initialise valid row counter

% Loop on row
for i = 1:DETECT_ZONE_HEIGHT
   % Research a valid strip
   % ----------------------
   pixel_nb = 0;                        % Initialise row white pixel counter
   % Loop on column
   for j = 1:DETECT_ZONE_WIDTH
      if im_diff(i,j)
         pixel_nb = pixel_nb + 1;       % Count white pixel
         col_sum(j) = col_sum(j) + 1;   % Memorize white pixel number per column
      end
   end
   % Count successive valid row
   if pixel_nb >= MIN_PIX_NB_PER_LINE
      row_nb =  row_nb + 1;
      if row_nb == 1
         begin_row = i;                 % Save first valid row index
      end
   else
      % End of strip detection
      % ----------------------
      if row_nb >= MIN_LINE_PER_STRIP
         % Research valid blobs
         % --------------------
```

```
        col_nb      = 0;    % Initialise valid column counter
        pixel_nb    = 0;    % Initialise blob pixel counter
        blob_nb     = 0;    % Initialise blob counter
        for j = 1:DETECT_ZONE_WIDTH
          if col_sum(j) >= MIN_PIX_NB_PER_COLUMN
            col_nb = col_nb + 1;              % Count column pixel
            pixel_nb = pixel_nb + col_sum(j);   % Memorize white pixel number
            if col_nb == 1
              begin_col = j;                % Save first valid column index
            end
          else
            % End of blob detection (column with none white pixel)
            if pixel_nb >= MIN_PIX_NB_PER_BLOB
              blob_nb = blob_nb + 1;          % Count valid blob number
              if blob_nb <= 2
                % Memorize blob center coordinates
                blob_data.x(blob_nb) = begin_col + (j-begin_col)/2;
                blob_data.y(blob_nb) = begin_row + (i-begin_row)/2;
                % COMMENT FOR C VERSION : eyes coordinates must
                % be rounded to the nearest integer
              end
            end
            col_nb = 0;     % Reset valid column counter
            pixel_nb = 0;   % Reset blob pixel counter
          end
        end
        if (blob_nb==2)
          % Check eye's space if only 2 blobs are detected
          blob_data.eyes_space = blob_data.x(2) - blob_data.x(1);
          if (blob_data.eyes_space<=MAX_EYES_SPACE) &&
(blob_data.eyes_space>=MIN_EYES_SPACE)
            if flag_ok==0
              flag_ok = 1;
            else
              % More than one strip valid --> Detection cancelled
              flag_ok = 0;
              break;
            end
          end
        end
      end
      row_nb  = 0;                      % Reset valid row counter
      col_sum = zeros(1,DETECT_ZONE_WIDTH);   % Reset vertical accumulation vector
    end
end
```

Face extraction and size normalization

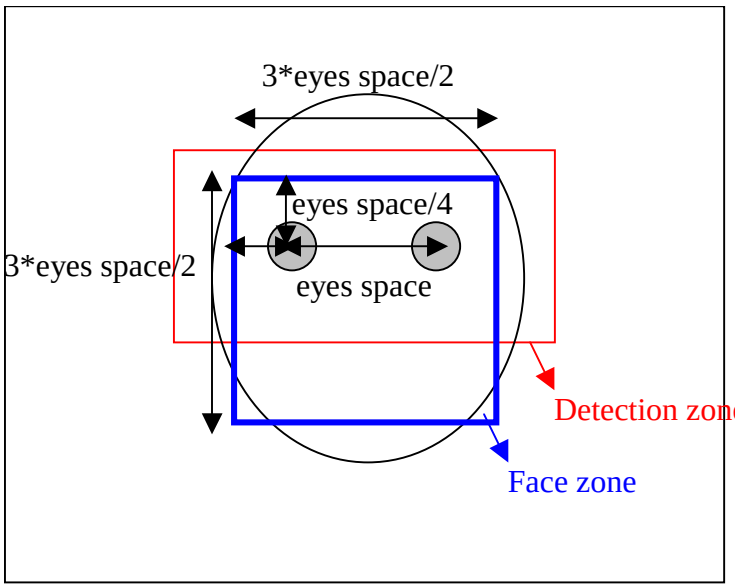
Objective and principle

When a blinking is detected, the GetFaceZone function must extract a part of the full current frame according to the position of the first detected eye and the calculated space between eyes. This part, called further face zone, is also normalized in size.

Face extraction

The eyes coordinates returned by the DoEyesBlinkingDetection function are relative to the detection zone. To extract the main part of the face, the first pixel coordinates and the size of the face zone are defined by the following equations:

usFaceZoneX0 = ausX[0] + usDetectZoneX0– usEyesSpace /4
usFaceZoneY0 = ausY[0]+ usDetectZoneY0– usEyesSpace /4
usFaceZoneSize = 3* usEyesSpace /2



Notes:

- Some controls have to be done to check that face zone parameters are inside the full frame.

Normalization in size

To be able to compare a face with another face, a first normalization in size is done to

obtain always the same normalized eyes space. This normalized eyes space must be equal

to the minimum available eyes space in order to simplify the size normalization step to only a sub-sampling operation (original size is always reduced, never extended).

As shown above, the face zone size is determined according to the detected eyes space:

usFaceZoneSize = 3* usEyesSpace /2

And the normalized face zone size is determined according to the configured minimum eyes space:

usNormFaceZoneSize = 3* usMinEyesSpace /2

C function specification

Prototype

```
char GetFaceZone(    unsigned char      *pucCurrentFrame,
                     unsigned char      *pucFaceZone,
                     t_sSizeNorm        *psAlgoParams )
```

The function must returns:

- -1 if an error has been detected
- 0 if the face zone is extracted and normalized in size.

Structure definition

Structure t_sSizeNorm definition:

Field	Type	Comments
usCurrentFrameHeight	unsigned short	Height of current frame (row number)
usCurrentFrameWidth	unsigned short	Width of current frame zone (column number)
usFaceZoneX0	unsigned short	First column of face zone
usFaceZoneY0	unsigned short	First row of face zone
usFaceZoneSize	unsigned short	Size of face zone
usNormFaceZoneSize	unsigned short	Size of face zone normalized in size

Input

Name	Type	Comments
pucCurrentFrame	Pointer on unsigned char	Pointer on the current frame buffer
psAlgoParams	Pointer on structure t_sFaceConf	Structure containing all eyes blinking detection parameters

Output

Name	Type	Comments
pucFaceZone	Pointer on unsigned char	Pointer on a buffer allocated for normalized face zone. Face zone is squared and width = height = 3/2 * the configured minimum valid eyes space.

Matlab sources

In matlab, ROI extraction and size normalization are done in two single instructions:

face_zone = frame(y0:(y0+height-1), x0:(x0+width-1));
norm_face_zone = imresize(face_zone, [norm_face_zone_size norm_face_zone_size], ‘bilinear’);

A sub-sampling algorithm can replace theses functions. A sub-sampling principle used in e2v sensor is given below. This algorithm calculates the next pixel address according to the selected sub-sampling factor. The same algorithm can be used to calculate the next row address.

The sub-sampling factor has to be calculated according to the current face zone size, the normalized face zone size and the sub-sampling precision (a sub-sampling precision of 1/64 is requested by the application):

Sub-sampling factor = (64*current face zone size) / normalized face zone size

```
%*****************************************************************
% Script : Sub-sampling
%*****************************************************************

SUB_STEP = 64;% Sub-sampling step = 1/64
SUB_FACTOR = 100;
INT_INC = floor(SUB_FACTOR/SUB_STEP)    % Integer increment
DEC_INC = SUB_FACTOR - SUB_STEP*INT_INC % Decimal increment
W_OUT_REQ = 60;                  % Requested final pixel numbers

%RESULT = [];
ADR_PIX = 0;
DEC_ACC = 0;                     % Decimal accumulator
COUNTER_PIX = 0;

while COUNTER_PIX < W_OUT_REQ

   COUNTER_PIX = COUNTER_PIX + 1;
   %RESULT = [ RESULT; ADR_PIX DEC_ACC COUNTER_PIX];

   DEC_ACC = DEC_ACC + DEC_INC;
   if DEC_ACC >= SUB_STEP
      ADR_PIX = ADR_PIX + INT_INC +  1;
      DEC_ACC = DEC_ACC -  SUB_STEP;
   else
      ADR_PIX = ADR_PIX + INT_INC;
   end

end
% RESULT    % display result for de
```

Contrast normalization

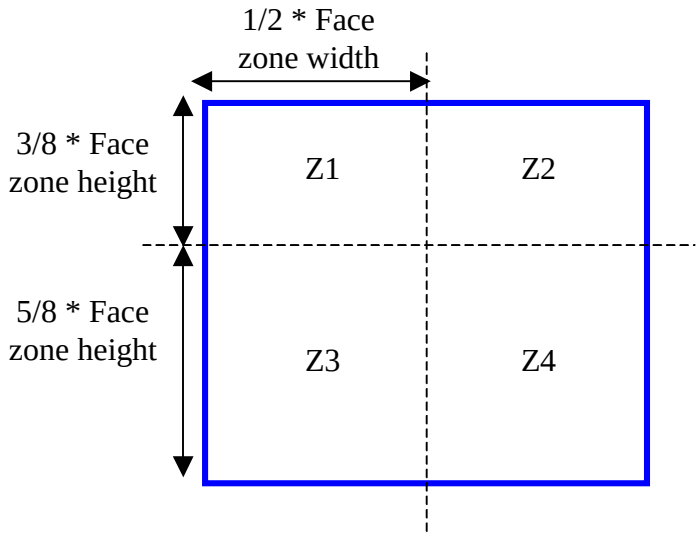
Objective and principle

The DoContrastNormalization function does a second normalization in contrast to reduce contrast differences due to environmental conditions.

To be more efficient, this normalization is done separately for different parts of the face.

The normalized face zone is divided in 4 zones:

- Left upper part containing the right eye
- Right upper part containing the left eye
- Left lower part containing the right part of the nose and of the mouth
- Right lower part containing the left part of the nose and of the mouth



Normalized face zone example

Notes:

- Normalized face zone is square, so width = height on the schema above.

The normalization in contrast for each zone includes the following steps:

1. Compute mean and estimation of standard deviation of each zone
   a. $m_k = \frac{\sum_{(i,j)\in Z_k} pixel(i,j)}{N_k}$ with N_k the number of pixel for each zone.

b. $\sigma_k = \frac{\sum_{(i,j)\in Z_k} abs(pixel(i,j) - m_k)}{N_k}$

2. Normalize the contrast of each zone

a. $pixel(i,j)_{(i,j)\in Z_k} = 128 + G * \frac{(pixel(i,j)_{(i,j)\in Z_k} - m_k)}{\sigma_k}$ with G a configurable gain.

Notes:

- Normalized pixel value must be bounded by 0 and 255.

C function specification

Prototype

```
char DoContrastNormalization(     unsigned char      *pucFaceZone,
                                  unsigned short     usFaceZoneSize
                                  unsigned char      ucContrast)
```

The function must returns:

- -1 if an error has been detected
- 0 if face zone has been normalized in contrast

Input

Name	Type	Comments
pucFaceZone	Pointer on unsigned char	Pointer on a frame buffer containing the face zone normalized in size.
usFaceZoneSize	unsigned short	Face zone size
ucContrast	unsigned char	Contrast factor

Output

Name	Type	Comments
pucFaceZone	Pointer on unsigned char	Face zone is normalized in contrast if return = 0.

Matlab sources

```
function [normalized_face] = DoContrastNormalization( face )

% Objective :   Normalize detected face in term of contrast
%
% Inputs :
%   face            detected face
%
% Outputs :
```

```
%   normalized_face     normalized face
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Mean normalization zone by zone
[nb_row, nb_col] = size(face);
z1 = face(1:floor(3*nb_row/8),1:floor(nb_col/2));
z2 = face(1:floor(3*nb_row/8),floor(nb_col/2)+1:end);
z3 = face(floor(3*nb_row/8)+1:end,1:floor(nb_col/2));
z4 = face(floor(3*nb_row/8)+1:end,floor(nb_col/2)+1:end);

z1 = (single(z1)-mean2(z1))/std2(z1);
z2 = (single(z2)-mean2(z2))/std2(z2);
z3 = (single(z3)-mean2(z3))/std2(z3);
z4 = (single(z4)-mean2(z4))/std2(z4);

gain_contrast = 20;
z1 = uint8(128 + gain_contrast*single(z1));
z2 = uint8(128 + gain_contrast*single(z2));
z3 = uint8(128 + gain_contrast*single(z3));
z4 = uint8(128 + gain_contrast*single(z4));

normalized_face = uint8(128*ones(size(face)));
normalized_face(1:floor(3*nb_row/8),1:floor(nb_col/2))          = z1;
normalized_face(1:floor(3*nb_row/8),floor(nb_col/2)+1:end)      = z2;
normalized_face(floor(3*nb_row/8)+1:end,1:floor(nb_col/2))      = z3;
normalized_face(floor(3*nb_row/8)+1:end,floor(nb_col/2)+1:end)  = z4;
```

Template creation

Objective and principle

The GetTemplate function must manage the template creation of a person during the

enrolment application.

The principle is:

1. The function must initialize the template when the first normalized face zone is selected.
2. The function must add the new selected normalized face zone on the current template as long as the number of selected face is smaller than a configurable number.
3. The function must create the template when the number of selected face is equal to the configured number (division of the current template by the configured number, the division can be done by a left shift operation, because the configured number will always be a power of 2).

Example of template

C function specification

Prototype

```
char GetTemplate(     unsigned char       *pucFaceZone,
                      unsigned short      *pusTemplate,
                    unsigned short usFaceZoneSize,
                      unsigned char       ucFaceNb,
                      unsigned char       ucInitRequest)
```

The function must returns:

- -1 if an error has been detected
- 0 if template has been updated (accumulation of the new face zone)
- 1 if template has been created

Input

Name	Type	Comments

pucFaceZone	Pointer on unsigned char	Pointer on a frame buffer containing the face zone normalized in size and in contrast.
pusTemplate	Pointer on unsigned short	Pointer on a frame buffer containing the current template in creation.
usFaceZoneSize	unsigned short	Face zone size
ucFaceNb	unsigned char	Number of normalized face zone requested to create a template = $2^{ucFaceNb}$
ucInitRequest	unsigned char	Initialization flag: = 1 for the first selected face zone = 0 in the other cases

Output

Name	Type	Comments
pusTemplate	Pointer on unsigned short	Current template is updated if return = 0. Current template is created and can be saved if return = 1.

Matlab sources

In matlab version, all selected normalized face zone are stored into bmp file. The template creation function reads again each file, sums all pictures and saves the mean picture into a bmp file.

Template comparison

Objective and principle

The DoRecognition function must compare a normalized face zone to a given template,

and indicates if the person is accepted or refused.

The function must calculate the distance between the given template and the normalized face zone. This distance is equal to the mean of the absolute value of the difference between each pixel of the normalized face and pixel of the template. The person is accepted only if this distance is smaller than a configurable threshold.

$$distance = \frac{\sum_{(i,j)} abs\left(norm\_face\_zone(i,j) - template(i,j)\right)}{\sum_{(i,j)}}$$

C function specification

Prototype

```
char DoRecognition(  unsigned char       *pucFaceZone,
                     unsigned char       *pucTemplate,
                     unsigned short usFaceZoneSize,
                     unsigned char       ucDistTh,
                     unsigned char       *pucDistance )
```

The function must returns:

- -1 if an error has been detected
- 0 if the person is refused
- 1 if the person is accepted

Input

Name	Type	Comments
pucFaceZone	Pointer on unsigned char	Pointer on a frame buffer containing a face zone normalized in size and in contrast.
pucTemplate	Pointer on unsigned char	Pointer on a frame buffer containing the selected template for the recognition application.
usFaceZoneSize	unsigned short	Face zone size
ucDistTh	unsigned char	Distance threshold to make recognition decision

Output

Name	Type	Comments
pucDistance	Pointer on unsigned char	The calculated distance between the selected template and the face zone.

Matlab sources

```
function [flag_ok, distance] = DoRecognition( face, template, threshold)

% Objective :   Compare the normalized face to the selected template
%
% Inputs :
%   face        Normalized face zone
%   template    Selected template
%   threshold   Maximum distance value to be accepted
%
% Outputs :
%   flag_ok     Flag =  1 --> person accepted
%                       0 --> person refused
%   distance    Distance between face and template
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize flag result
flag_ok = 0;

% Compute distance
distance = abs(face - template);

if distance <= threshold
   flag_ok = 1;
end
```

Development constraints

Environment tools

The development is done under Linux Environment.
The development tools are generated from the Buildroot supplied by e2v
The embedded Kernel (Linux) is also supplied by e2v.

The platform user guide describes how to generate the builroot, the Root File System and the kernel

This development is made only in user space, so the kernel or the buildroot must not be change

Development rules

The development must follow the e2v coding rules.
This development has the following constraint:

- Some processing functions are made in “real time”. Each time a picture is available, the processing must be done before the over picture (15 frame per second)
- Don’t used float, double
- Avoid mathematic functions if possible
- Optimization of loops

Function must be validated on the e2v platform.
To check the intermediate results for each function, e2v purchases picture or video for input, and the Matlab results after each function.