The Speech Video Acquisition and Analysis Modules of the ABCP1 System

Technical Report TR-ABCP1-07

Vitor M M C Pera

FEUP - Porto
July 2014

Abstract

This report describes in detail the Speech Video Acquisition and Analysis Module (SVAAM) that was built to integrate the ABCP-1 system. If some parts of the report are more detailed than it would be appropriate, that was intentional and is also related to the author learning method. Although the SVAAM baseline version accomplished the established requirements, the report presents a few suggestions that likely can improve it. The code used to implement the SVAAM baseline version, based on a freely available (GNU-licence) image processing library, is also included in this report.

Contents

1 Introduction

Generally, using two pieces of information does not grants that their contributions "add" improving the decisions, indeed "simply adding" clues may even become catastrophic. Moreover, in the AVSR field often the contributions of the acoustic and of the visual feature streams are very unbalanced, not only when the acoustic environment is clean, in terms of the amount of (discriminative) speech information they convey. The visual stream usually is weaker, although when the acoustic conditions degrades too much then eventually the visual stream becomes dominant. Anyhow, the referred usual unbalance reinforces the claim to extract good - as good as possible - visual speech features. This exigence implies natural difficulties, which in this particular work were addressed keeping in mind two main points. One consists of setting a good compromise between designing a user-friendly interface and, by the other side, imposing some restrictions in order to make the SVAAM task affordable. The other point consists essentially of the reassurance to use *state of the art techniques*, benefiting from public and freely available (under a GNU-licence) functional modules developed by experts on image processing [1]. The SVAAM sub-system was also built with the purpose to use it later, as a baseline development platform, in more demanding applications in terms of the visual speech features extraction. Noticing this may help to understand why some approaches were followed to deal with concrete problems, in some cases simpler solutions could have been implemented.

The structure of this report is as follows. Section 2 presents the application interface and a few aspects about the video acquisition. Section 3 presents the implemented Region of Interest (ROI) detection and ROI tracking mechanisms. Section 4 is dedicated to the features analysis method. In the Section 5 are presented a few suggestions for improving the SVAAM module regarding to the ROI tracking robustness and speedup, and also to the features pos-processing. The last subsection addresses some *Nonlinear mapping* approaches using graph-based methods, which eventually could afford a good framework for developing the front-end module of AVSR systems. Final conclusions are drawn in Section 6.

2 The interface

The speech interface must comply with a realistic use in a office-like environment. It must not be required special illumination of the user face, the natural and/or artificial illumination in a normal office room must satisfy completely. It is required that the position of the speaker in relation to the

[1] In particular, the *OpenCV* library (http://opencv.org), which is at the basis of the implemented SVAAM module.

video camera does not implies unnatural static poses. That is, most of the common movements when using a desktop PC, for instance, must be possible without compromising the efficiency of visual speech features. Better than words, a few images certainly may enlight what is expected from the ABCP1 interface concerning to the image stream. Figure 1 shows a sequence of frames extracted at regular intervals (approximately 1 second sampling rate) from a video file [2] corresponding to a typical use of the interface. It

Figure 1: Sequence of frames (approximately 1 second interval between consecutive photograms) from a typical video captured by the ABCP1 interface.

must be referred that in this particular video the SVAAM module was able to extract correctly the visual speech features. In this particular video the illumination was natural, from a window almost frontal to the user face. It can be seen that the camera (that was fixed with a tripod, framing part of

[2]This video file, that was collected for the training data to develop the ABCP1 system, is available in the address http://speech-rec-vcp.com; that video also show the ROI sequence computed by the SVAAM module.

the upper body) is approximately at the eyes level, though with significant tolerance (this video was recorded with the speaker sitting in a normal chair with rollers and variable inclination back). It is expected that the face is not obstructed by the microphone or other object(s), such as illustrated. No special facial makeup, such as lip-marking, is required. It can also be observed that the interface must be very tolerant to the user movements along the camera axis. For instance, the size of the face in the frame number 7 is approximately half of the size in the last frame. That tolerance requires the abilities to capture ROIs with quite different size - it must be noticed that in the case of the frame 7 the size of the ROI, in pixels, may be quite small if the camera resolution is small - and also to perform some normalization of it. Tolerance must also exist in relation to movements over planes orthogonal to the camera axis, just requiring that the face keeps inside the frame. The frames 7 and 9 illustrate the interface ability to allow some head inclination, being able to "correct" those occurrences in order to avoid affecting the visual features. However, it must be stressed that is not expected that the interface presents special tolerance to rotation movements of the head in relation to other axis (this decision was taken as a designing compromise such as explained in the Section 1). A relevant requisite is that no special background is needed (backgrounds in typical office rooms must present no troubles). The strip-film also illustrates (the referred video does it clearly) the interface requisite to cope with natural movement in terms of velocity, though the tolerance to brisk movements is not required.

In relation to the video capturing device, a standard non-professional, *OpenCV-compatible*, video camera providing compressed video signal must satisfy. In the particular case of the collected training material, were recorded MPEG-2 compressed files with the specifications: 720 *pixels* x 576 *lines*, *interlaced video*, 25 *fps* and YUV420P *pixel format*.

3 The ROI detection and tracking

3.1 Introduction

Besides satisfying the presented interfacing requisites, the SVAAM module must also comply with an essential requirement: it must not prevent the ABCP1 recognizer to operate in real time.

Broadly, the SVAAM specifications were pursued properly combining powerful image processing algorithms from the *OpenCV* library with appropriate geometric normalization techniques that reduce the features intra-class variability associated to some particular movements.

Although in the SVAAM existing version the ROI consists simply of the mouth region, other facial marks are captured. Two main reasons were at the basis of this approach. One is related to the fact that the mouth can be relatively difficult to detect and track. For example, the localization of

the nostrils often is more reliable comparing to the mouth and probably this difference becomes even larger if, for instance, the image is affected by shadows. It must be noticed that the mouth region presents particular difficulties also due to the variable configuration of the lips during speech. Moreover, in principle redundancy makes the ROI determination more robust if multiple facial marks are considered. The other relevant reason is based on the possibility that the auxiliary facial marks offer to normalize the ROI in relation to some head movements, using very simple geometric techniques. Therefore, the noise affecting the visual features and due to those movements could be effectively reduced.

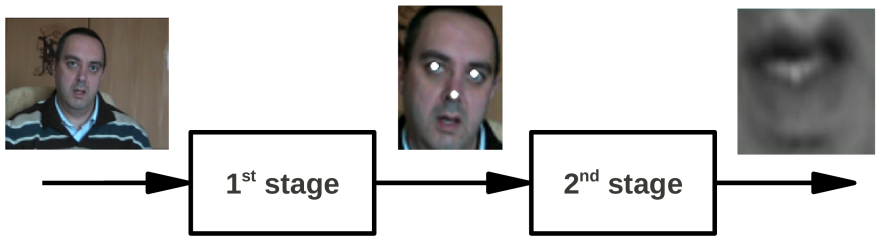

Figure 2: Two stages approach to determine the ROI: 1) auxiliary marks are localized first, allowing to determine approximately the ROI (eventually after performing some normalization); 2) from that result, the ROI is then accurately fixed.

In the initial frame the ROI is localized following the steps: 1) face detection, using a *Boosted Cascade Classifier*[8] (BCC), then 2) auxiliary facial marks detection (also through the BCC), and finally 3) approximate ROI localization using simple geometric relations (since no visual features are extracted from the initial frame, an accurate localization was not implemented). After detection, for each new frame a tracking procedure follows: 1) the new localizations of the auxiliary marks are determined running the BCC over the face regions determined from the respective previous localizations, then 2) the ROI is approximately determined based on geometric relations, and finally 3) running the precise *Template Matching* (TM) algorithm (in the *OpenCV*) the ROI is accurately determined.

The BCC and the TM classifiers above referred were combined trying to exploit theirs relative strengths, in terms of speed and accuracy. In general the BCC is faster than the TM and in this application in particular is accurate enough even configuring it to favour speed. So, the BCC was used in this system to detect and track the auxiliary facial marks, allowing real time operation of the SVAAM module in a *standard PC* (2012). Another key aspect is the possibility to exploit the inherent concurrency of the tasks consisting of the detection and tracking of the auxiliary marks to speedup the computations (see Section 5.3). By the other side, the TM can be very

accurate but is relatively slow. So, initially the searching region for the ROI is shorten using the BCC algorithm and the known auxiliary marks positions, setting a well conditioned problem, and only then the TM is used to accurately determine the ROI. It is important to recall that the ABCP1 is intended to be a speaker dependent recognizer, which in principle renders more efficient and reliable the TM based approach. Section 5.2 presents a few suggestions in order to improve the overall robustness.

In the next two subsections, the ROI (and auxiliary facial marks) detection and tracking implemented procedures are described in more detail.

3.2 The ROI detection

Follows the pseudo-code corresponding to the implemented baseline version of the auxiliary face marks and the ROI detectors. The C++ respective code can be found in the Appendix A.

Algorithm 1 ROI detector (baseline version)

Require: *videostream*; BCCs for the Face, Nose, LeftEye and RightEye
Ensure: ROI, $leftEyePos$, $rightEyePos$, $nosePos$

1: **repeat**
2: $frame \Leftarrow grabFrame(videostream)$
3: $faceRect \Leftarrow detectFace(frame)$
4: **until** faceIsReady($faceRect$)
5: $leftEyePos \Leftarrow detectLeftEye(faceRect)$
6: $rightEyePos \Leftarrow detectRightEye(faceRect)$
7: $nosePos \Leftarrow detectNose(faceRect)$
8: $ROI \Leftarrow detectRoi(leftEyePos, rightEyePos, nosePos)$
9: **return** ROI; auxiliary face marks $leftEyePos$, $rightEyePos$ and $nosePos$

The cycle through the lines 2-3 was established considering that at the beginning of any recognition session the user eventually is not properly in front of the camera as needed. The face detection (line 3) is performed using a Boosted Cascade Classifier (BCC) using Haar-like features, which is available jointly with the OpenCV-2.3.1 package in the *SourceForge* repository[3]. The parameters used to control the conditions established to begin any session can be easily set manually.

Each detection procedure in the lines 5-7 is also based on a BCC corresponding to the respective facial region[4]. The determined localisations of the auxiliary marks are used in the initial iteration of the *ROI tracking*. Since this detection code runs only once, it was not tried to accelerate it.

[3] http://sourceforge.net/
[4] http://note.sonots.com/SciSoftware.html

Concerning to the robustness, in the case of the eyes it was already developed some preliminary work for training new BCCs with the existing dataset specific to this application and subject.

In the line 8 the ROI is determined using the available coordinates of the auxiliary face marks and simple geometric relations based on a small set of parameters, that are easily tuned for the (single) user of the application. In the actual implementation of the detector the ROI is only needed to extract information to adapt parameters to be used in a later optional procedure, not requiring an accurate localization, so the procedure finishes at this point.

According to line 9, the estimated ROI and the positions of the auxiliary facial marks are made available for the tracking procedure.

3.3 The ROI tracking

Next is presented the pseudo-code corresponding to the ROI tracking implemented baseline version. In the Appendix A is available the respective C++ code.

Algorithm 2 ROI tracker (baseline version)

Require: *videostream*; BCCs for the Nose, LeftEye and RightEye; Lips Template; ROI detector outputs $leftEyePos$, $rightEyePos$ and $nosePos$

Ensure: ROI

1: **repeat**
2: $frame \Leftarrow grabFrame(videostream)$
3: $nosePos \Leftarrow trackNose(nosePos, frame)$
4: $leftEyePos \Leftarrow trackLeftEye(leftEyePos, frame)$
5: $rightEyePos \Leftarrow trackRightEye(rightEyePos, frame)$
6: $approxROI \Leftarrow approxLoc(nosePos, leftEyePos, rightEyePos, frame)$
7: $ROI \Leftarrow tuneLoc(approxROI)$
8: **until** ends(video stream)
9: **return** ROI

The code structure is very simple, with the lines 3 to 5 tracking the auxiliary face marks which estimated coordinates are used to approximately find the ROI, in line 6, and afterwards (line 7) accurately determine the ROI.

Each one of the procedures performed in the lines 3 to 5 is based on the BCC, for the respective facial mark, already used in the ROI detector. It was found that the position of each auxiliary mark could be well determined, in terms of robustness and accuracy, and also quickly enough, using this classification method over relatively small regions. For each face mark a simple mechanism was implemented to reduce the searching region, based on the related information from the previous iteration. It was assumed that

wide and brisk head movements do not occur, and the parameters were set running several experiments with the subject moving the head normally (in fact, more freely than it would be normal in this kind of applications). More complex approaches, such as modeling more precisely the trajectory of the facial marks, were not necessary since the implemented solution turned out to be completely satisfactory. In the current version, each auxiliary mark is processed separately from the other ones, though the robustness could possibly increase using, for instance, the output or partial results obtained processing one of the marks to assist in the detection of the others. In principle, the existing redundancy in the extracted information could be useful to recover from (eventually catastrophic) mistrackings already at this stage. However, remains the possibility to do that later (in the *approxLoc*() function), at least in some cases. For each facial auxiliary mark the OpenCV function *cascade.detectMultiScale*(), that calls the BCC, is configured in a way that can return more than one hypothetical region. Therefore, the Hausdorff spatial similarity function[5], which has properties that are appropriate for this task, is used to select one of those regions.

In the actual version, the *approxLoc*() function (line 6) performs normalisation of the mouth region, based on the coordinates of the auxiliary marks, in relation to some head movements. Eventually, this includes scaling and image transformation corresponding to head rotation. It must be emphasized that the ABCP1 is intended to be a speaker dependent system, so not requiring any user normalisation mechanism. To achieve the ROI normalisation the baseline version approach is quite simple, avoiding geometric transformations based on the auxiliary marks estimated coordinates in a 3D facial model. Nevertheless, the implemented approximation leads to quite good results in relation to the head rotation around the X-axis (Figure 3). The small set of parameters used in these geometric transformations can be easily tuned.

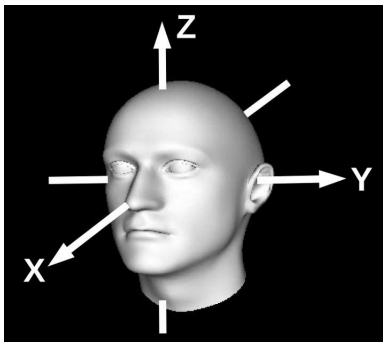

Figure 3: Head's axis.

[5] http://www.alexandria.ucsb.edu/archive/2003/similarity.html

The *tuneLoc*() function (line 7) calls the OpenCV *matchTemplate*() function that applies the Template Matching (TM) method to the input (approximate) ROI and returns the final *ROI*. This method performs very robustly and accurately, such as expected given its records and the fact that the application is speaker dependent and some geometric normalization was already executed. Although the TM is known to be relatively slow, not at all that became a problem in this module because of the implemented mechanisms to reduce the searching region.

4 The features extraction

4.1 Introduction

The features to represent visual speech information have been categorized as: geometric features; model based features; visual motion features; or image based features.

In general, the model based features are robust, compact and perform some face normalization imposing a visual structure that automatically reduces unwanted variability of the feature vectors. These properties justify that methods such as the Active Appearence Model (AAM) are often used in AVSR systems. The visual motion features are based on the assumption that the motion of the visible speech articulators conveys relevant speech information and also that its "dynamic" nature automatically eliminates part of the variability associated to non-speech information.

In spite of the merits of these two categories of features, in the SVAAM the choice was different, according to the following arguments. The fact that the SVAAM integrates a speaker dependent recognizer was decisive to favour image based features, since the larger complexity of other approaches, allowing to improve the multi-user (-face) robustness, does not brings any advantage in practice. These approaches may present another relevant disadvantage in relation to image based features such as the Bi-dimensional Discrete Cosine Transform (DCT) used in the SVAAM, which is related to the likely over-simplistic assumptions at their basis. For instance, in general the AAM based features (often regarded as one of the most capable approaches in AVSR) do not capture (model) the position neither of the teeth nor the tongue. Contrarily, using the DCT often becomes possible (if enough coefficients are preserved) to discriminate among consonants such as /s/, /sh/ and /f/ (or /z/, /zh/ and /v/), or between vowels such as /iy/ and /ae/ [6], based on the image captured by those articulators (obviously, those phonemes correspond to articulation places in the front of the oral cavity). Moreover, in some circumstances the discriminative ability of image based

[6]Examples of words using the phonemes: /f/ in "fair"; /s/ in "sad"; /sh/ in "she"; /v/ in "vim"; /z/ in "zoo"; /zh/ in "joy"; /iy/ in "feel"; /ae/ in "at".

features significantly benefits from specific facial marks (in the ROI) that are not captured by other approaches. The image based features, and the DCT is no exception, are in general robust to ROI localization inaccuracies, which obviously is desirable even if other (but not all) feature categories also have this quality and if the ROI tracking mechanism is quite reliable. The compression ability of the DCT is satisfactory in the particular case of the SVAAM (more details can be found in the Section 4.2), although other analysis methods (e.g. the AAM) could lead to more compact representations. Generally, image based features are straightforward, do not require modelling assumptions beyond those implicit in their (solid) mathematical foundations (still allowing engineering intuition, such as shown in Section 4.2), and are easy to code. The fact that fast algorithms (and even dedicated hardware) have been developed for image based features such as the DCT obviously contributes to their choice.

In the SVAAM, the image based features (DCT coefficients) are combined with geometric features, simply concatenating the respective feature vectors. Geometric based features are often regarded with some suspicion but in the particular case of the SVAAM two key aspects were decisive to include them. One aspect is that, besides the normal usage, these features can be specially appropriate to model deterministic (hard) dependencies. For instance, in the case the ABCP1 system based on a Dynamic Bayesian Network it is straightforward including valuable (ingenious) deterministic relations between (statistical) variables according to the observations of geometric features as simple as the "lips height" or the "lips width", just to give an example. The other argument in favour of the geometric features is that it could not be more easy to extract some basic though useful geometric features, namely the referred "lips height" and the "lips width"; they are directly extracted from the ROI rectangle.

4.2 The DCT based analysis

The DCT was selected as the image compression transform mainly because[2]: 1) its information packing ability is superior to other important transforms, such as the Discrete Fourier Transform (DFT), though not being optimal (the Karhunen-Loève Transform (KLT) is optimal in terms of information packing); 2) it is data independent (such as the DCT, for instance, but contrarily to the KLT); 3) computationally is less complex than other transformations (e.g. KLT); 4) in the case of using sub-images, the DCT eventually is advantageous in relation to other transforms (DFT including) since the reconstructed image has not a block-like appearance (due to the general Gibbs phenomenon associated to the the truncation of the summations, affecting DCT and other transforms); 5) being a good compromise between information packing ability and computational complexity, it is an image compression standard (and it is implemented in the *OpenCV* platform).

Given and image $I(x, y)$ with size NxN, the (forward) DCT is

$$DCT(u, v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I(x, y)g(x, u)g(y, v), \quad u, v = 0, 1, ..., N-1 \quad (1)$$

with

$$g(r, s) = \alpha(r) \cos \frac{(2s+1)r\pi}{2N} \quad (2)$$

where

$$\alpha(0) = \sqrt{\frac{1}{N}} \texttt{ and } \alpha(r) = \sqrt{\frac{2}{N}} \texttt{ if } r > 0 \quad (3)$$

The size of the image submitted to the DCT, after the ROI geometric normalization, is 32x32 pixels. The DCT is applied to the whole image, not following any sub-images based approach (for instance, 16 8x8 pixels sub-images). The fact that the ROI presents a strong symmetry in relation to the middle vertical axis was not used directly, but it was considered when selecting the transform coefficients (details are presented bellow).

Figure 4 shows the DCT coefficients extracted from a typical ROI image. According to the common results in natural (smooth) images, the

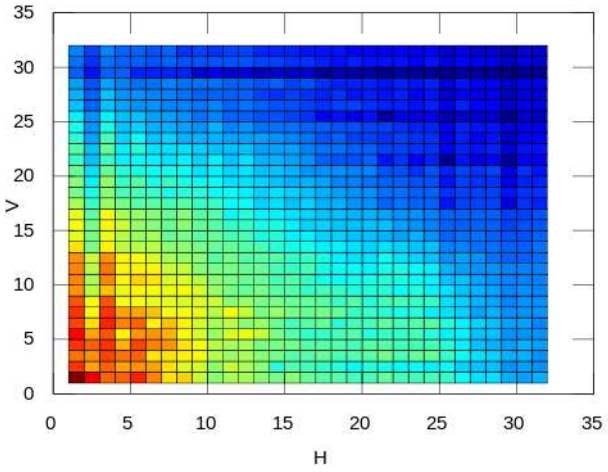

Figure 4: DCT coefficients (after log transform and using common color code (reddish means higher values)).

most relevant transform coefficients are located just around the origin of the image transform (notice that the figure represents the logarithm of the coefficients values, if linear scaling was used then the coefficients distribution in the spacial frequencies plane would form a very sharp peak near the origin). The figure reveals other interesting aspects. The coefficients along

the vertical direction are relatively more relevant, since the ROI in general presents a "richer structure" along that direction. It is important to preserve this tendency when choosing the coefficients since the speech visual realization "along" the vertical axis is more discriminative than "along" the horizontal axis. Another interesting aspect is that along the horizontal axis the lower order coefficients do not present a monotonous behavior, clearly in the second column the coefficients are smaller than in the third column. This reflects the ROI symmetry, such as the face, in relation to the vertical axis in the sagittal plane.

The selection criteria used to create a mask for the DCT coefficients was based on the assumption that the "coefficients variability" relates positively and strongly to the "speech information" (according to the information theory principle that information means uncertainty). Figures 5-7 present three masks with different sizes. In the first one are truncated $97,7\%$ of the coefficients, leading to a 24-dimensional features space. Even if this mask preserves $65,1\%$ of the variability (reconstruction $ems = 34,9\%_r$), considering the distribution of all the coefficients, naturally it can be expected that such a strong compression causes significant loss of speech information. To have some intuition about that loss, in Figure 8 are presented the reconstructed images (in the bottom row for this 24-size mask) for five typical lips configurations (in the top row are the original 32x32 images). It becomes clear that the discrimination ability is strongly preserved, tough naturally heavy loss of detail occurs. Figure 6 corresponds to truncating $96,4\%$ of the

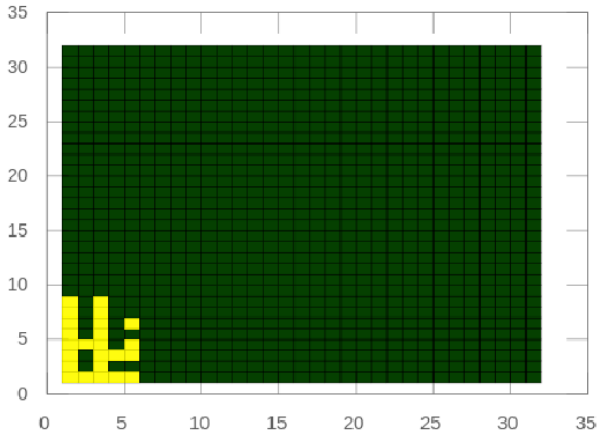

Figure 5: DCT mask obtained considering the 24 most variable coefficients.

coefficients leading to a 37-dimensional features space, preserving $74,9\%$ of the global variability (reconstruction $ems = 25,1\%_r$). Figure 8 shows the reconstructed images (in the second row, upwards). Although comparing with the previous mask the differences seem minor, previous experience suggests

that for developing the SVAAM, considering the ABCP1 specifications and the available training resources, this mask sets a better compromise between model accuracy and training robustness. Figure 7 corresponds to truncat-

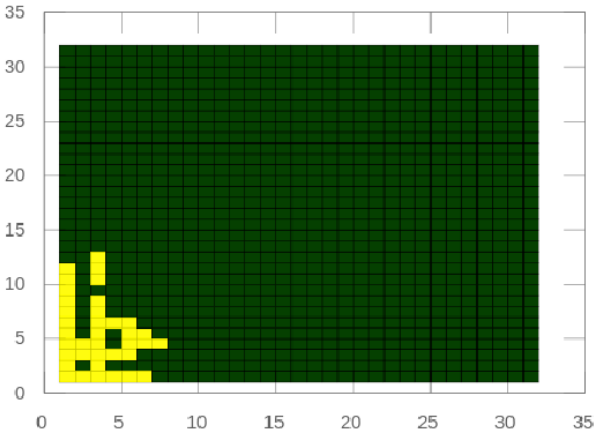

Figure 6: DCT mask obtained considering the 37 most variable coefficients.

ing $91,5\%$ of the coefficients leading to a 97-dimensional features space, preserving $90,0\%$ of the global variability (reconstruction $ems = 10,0\%_r$). Figure 8 shows the reconstructed images (in the third row, upwards), making clear that this mask preserves most of the visible original images details. Nevertheless, most of that detail conveys just residual speech information and such a high-dimensional features (which already are quite decorrelated) space dimension compromises the training robustness (curse of dimensionality problem).

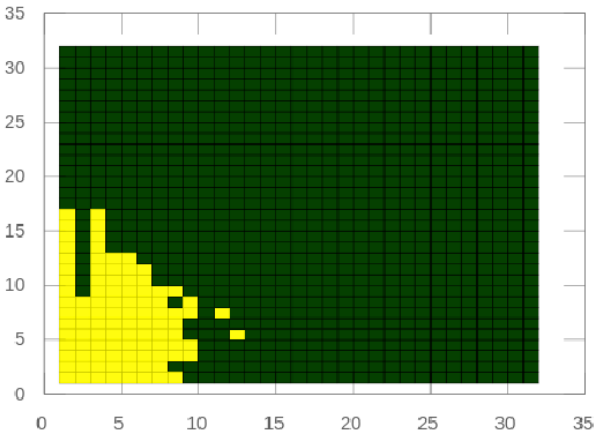

Figure 7: DCT mask obtained considering the 97 most variable coefficients.

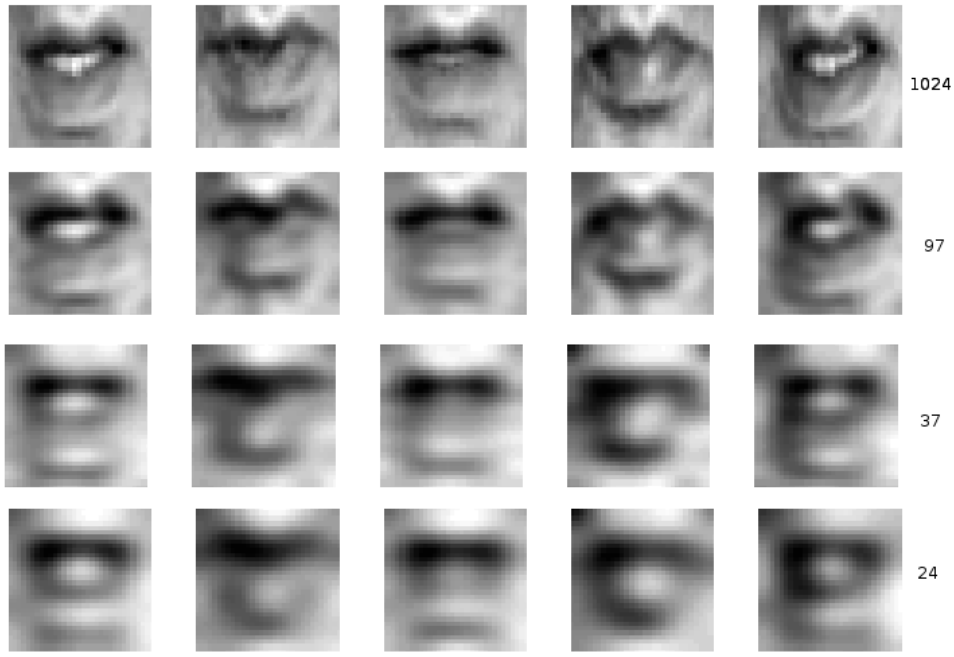

Figure 8: Original images (first row) and reconstructed images using different masks (size at the rightmost column).

4.3 The geometric features

In the Section 4.1 were already presented the two main arguments that supported the decision to include geometric features in the SVAAM. Indeed, only two features are used, the "lips height" and the "lips width". Other geometric features often used, such as the "lips area", were not used since it was intended not to make any effort to get those features (obviously, both the "lips height" and the "lips width" can be directly measured in the ROI rectangle). Even using just these two features, in many useful linguistic contexts they convey information that effectively contributes to discriminate among some classes of sounds or silence segments. The "hard" nature of these two features also contributed to choose them, due to modelling possibilities they offer (and that in the case of using the GMTK can be implemented very easily), such as sketched in the Section 4.1.

In the SVAAM these two geometric features are simply concatenated with the DCT coefficients, leading to a 39-dimensional features space (if using 37 DCT coefficients, obviously).

5 Future work

5.1 Introduction

The SVAAM version that was built completely satisfies the defined requirements, that is, following well known state of the art techniques: from a regular use of the interface (see Section 2), it is able to detect and track the ROI with enough accuracy (Section 3), allowing to deliver a feature vectors stream in real-time carrying relevant speech information (Section 4).

Said that, for sure space exists to improve the SVAAM. Next are presented four lines that likely lead to effective improvement, hopefully allowing to cope with more demanding applications. Increasing the robustness of the ROI localization and speeding up the respective computations are two of those lines. Other concerns to adding some dynamic information to the extracted feature vectors. A quite different approach consists of using some graph-based methods at the basis of the front-end module.

5.2 Increasing the ROI robustness

At least for the designed application and the collected data, the SVAAM module exhibited enough robustness, being able to detect and track the ROI accurately. But if the challenges to capture the ROI from the input video become substantially more demanding, for instance if the illumination strongly deteriorates or if too pronounced head rotating movements occur, then some improvements become necessary. Next are presented a few weaknesses that were identified and that could have real impact if the conditions become more difficult. In some cases a few suggestions are presented to cope with those difficulties, although the effectiveness of those ideas could not be verified yet (hopefully, in a near future that will be possible).

The detection of the nose using the BCC method (in the Algorithm 2) exhibited excellent robustness. But in the case of the eyes some difficulties could be felt, specially during the eye(s) blinking. Likely, any of the two following approaches, both quite straightforward, would be able to solve this problem. One consists of training BCCs for the eyes using specific data and including "eyes blinking" segments, instead of the actual classifiers that were just downloaded from the Web address http://sourceforge.net (personally, this solution is very attractive since the BCCs are a powerful technique and at least apparently not too much effort is needed to train them [7]). It worth's to remind that the ABCP1 recognizer is speaker dependent, and in this context is very important to take advantage of that. Another aspect related with the BCCs, eventually less relevant, is that only Haar-like features were used, opening the possibility to try obtaining some improvement using other features.

[7] http://note.sonots.com/SciSoftware.html#zce04ba262f2df9672c526e914250814f

The other approach arises from the fact that the implemented tracking mechanism, which is very simple, does not exploits the redundancy inherent to the simultaneous processing of the multiple (three) facial marks. Indeed, it is easy to implement mechanisms, some of them quite obvious, that may assist the tracking of any facial mark, based on information about the "recent trajectory" of other marks. For instance, if the eyes "go astray" then using information from the nose trajectory can be decisive to correct that situation.

Another problem is related with the occurrence of ample rotation of the head around not necessarily fixed axis. Actually, the SVAAM considers only the rotation along the X-axis (Figure 3), and no additional effort was made to build a more capable system (since it was not necessary for the used application data). Nevertheless, tracking algorithms requiring $3D$ object modeling techniques among others, may be needed in those situations. It must be emphasized, once again, the speaker dependency nature of the application, allowing to refine some techniques and tune the parameters.

5.3 Speeding up the ROI localization

The SVAAM efficiency, in terms of the computational effort, is critical since this module is a potential bottleneck preventing that the recognizer may operate in real time.

Two main approaches are planned in order to try speeding up the SVAAM. One is related to the fact that the BCC and also the TM classifiers are responsible for almost the entire computational effort to process each frame (the BCC runs three times and the TM runs only once, each frame). In spite of the efficiency of the BCCs, certainly would be advantageous to reduce further the evaluation time. Since the time efficiency is critical for many applications, in the literature can be found specific techniques addressing this issue[14]. In relation to the TM method, the size of the searching region affects strongly the computational cost. The existing SVAAM baseline version can still be improved taking advantage of the speaker dependency nature of the application.

The other approach is based on the fact that some of the costly operations may be carried out in parallel. At least considering the actual implementation, the tracking of the eyes and of the nose may be done simultaneously. Increasing the algorithms complexity, eventually other possible concurrent time consuming operations may be found. Moreover, since multicore computers are common nowadays, using for instance the OpenMP [8] application programming interface, becomes quite easy, just requiring very little changes in the code, to reduce substantially the time needed for tracking the ROI.

[8] http://www.openmp.org

5.4 Pos-processing the extracted features

The purpose of this pos-processing module is to add some dynamic information to the feature vectors. The merit of this approach has been verified empirically, besides being conceptually appealing (a similar approach is followed with the acoustic features stream).

To achieve this goal it is suggested, for the time being, to follow well-known and straightforward techniques which have been applied with success in several AVSR systems[1]. Basically, two main steps are needed. In the first step consecutive feature vectors are combined to compose an unique vector, which in principle conveys some dynamic information about the speech visual realization. Then the second step compresses that speech representation, trying to achieve a good compromise between the loss of information and the features space dimension.

In relation to the first step, usually is considered a time shifting window, with the same time-step than the original features stream, and the vectors inside that window are simply concatenated. Typically the dimension of the resulting vector becomes too high to allow the robust training of the models, justifying the second step.

In relation to the second step, often linear transformations as simple as the Principal Component Analysis (PCA) are used to reduce the dimensionality of the features space[15]. If enough labelled data exists (the elementary visual speech units usually are the visemes, but "visyllables" or some "automatic" sub-word units may be considered too), then likely a better approach is to use that information to try to determine the subspace (or the transformation) that allows the most discriminant representation (after projecting the feature vectors resulting from the first step). Quite a lot techniques have been proposed to find the "most discriminative directions" for problems like this one. As an example, the Fisher's linear discriminant, that is very easy to implement, can be a good option to begin some experimental work in this issue.

5.5 Alternative front-ends

Recently have been reported results on *nonlinear mapping*[16] suggesting that it can be very interesting to use some graph-based methods[17, 18] at the basis of the front-end module in AVSR systems. These techniques can be applied directly to the input images, so not requiring any previous features analysis stage. For instance, in the case of using 32x32 *pixels* then the input vectors would have dimension 1024. The dimensionality of the representation is reduced trying to preserve some properties of the local structure of the vectors distribution, in an unsupervised manner, exploiting the existence of manifold structures in those distributions. Indeed, results in *speechreading* (presenting exactly the same problem) have been reported

showing the remarkable potential of several non-linear algorithms presenting extraordinary compression ratios (in the order of the hundreds), and still preserving in great part the ability to discriminate among linguistic classes at the sub-word level. Moreover, the effort to carry on with the experimental work has been greatly reduced since tools implementing those methods have been made available freely. Hopefully in a near future, some research work will be pursued on this signal processing topic, which seems very promising for the audio-visual speech recognition technology.

6 Conclusions

The main goal of the work here reported, consisting of building a baseline version for the Speech Video Acquisition and Analysis Module of the ABCP1 recognizer, was fully achieved. Running the interface according to the specifications makes that the SVAAM module delivers, in real time, a stream of feature vectors conveying relevant visual speech information. The SVAAM module was built based on general problem solving approaches for combining *state of the art* image processing algorithms, available in the *OpenCV* library.

This work also embraced some examination of approaches for increasing the robustness of the ROI localization, allowing to keep the performance in more difficult tasks. The acceleration of the SVAAM computations and the improvement of the speech representation by means of features posprocessing techniques were other two promising developments addressed in this work and here briefly reported. *Nonlinear mapping* approaches using some graph-based methods was another topic that was addressed, resting the confidence of its great potential to use in the front-end module of AVSR systems.

References

[1] G. Potamianos, C. Neti, J. Luettin, and I. Matthews, Audio-Visual Automatic Speech Recognition: An Overview, Chapter of book "Visual and Audio-Visual Speech Processing", MIT Press, 2004.

[2] R.C. Gonnzales, and R.E. Woods, Digital Image Processing, Prentice Hall, 2001.

[3] D.A. Forsyth, and J. Ponce, Computer Vision: A Modern Approach, Prentice Hall, 2003.

[4] A.K. Jan, Fundamentals of Digital Image Processing, Prentice Hall, 1989.

[5] M. Sonka, V. Hlavac, R. Boyle, Image Processing, Analysis and Machine Vision, Thomson, 2008.

[6] R. Kaucic, and A. Blake, Accurate, Real-Time, Unadorned Lip Tracking, International Conference on Computer Vision, pp.370-375, 1998.

[7] M.C. Santana, O.D. Suarez, L.A. Canalis, and J.L Navarro, Face and Facial Feature Detection Evaluation: Performance Evaluation of Public Domain Haar Detectors for Face and Facial Fature Detection.

[8] P.I. Wilson, and J. Fernandez, Facial Feature Detection Using Haar Classifiers, Journal of Computing Sciences in Colleges, pp.127-133, 2006.

[9] R. Lienhart, and J. Maydt, An Extended Set of Haar-like Features for Rapid Object Detection, Journal IEEE ICIP, 2002.

[10] M. Gurban, J.-P. Thiran, T. Drugman, and T. Dutoit, Dynamic Modality Weighting for Multi-Stream HMMs in Audio-Visual Speech Recognition, International Conference on Multimodal Interfaces, pp.237-240, 2008.

[11] M. Yazdi, M. Seyfi, A. Rafati, and M. Asadi, Real Lip Contour Tracking for Audio-Visual Speech Recognition Applications, International Journal of Biological & Medical Sciences, Vol.4, Iss.4, p.190, 2009.

[12] P. Lucey, T. Martin, and S. Sridhran, Confusability of Phonemes Grouped According to their Viseme Classes in Noisy Environments, International Conference on Speech Science and Technology, Sydney, 2004.

[13] Calvin Lin, and Lawrence Snyder, Principles of Parallel Programming, Pearson/ Addison Wesley Ed., 2008.

[14] Tae-Kyun. Kim, and Roberto Cipolla, Multiple Classifiers Boosting and Tree-Structured Classifiers, Machine Learning for Computer Vision, SCI 411, pp.163-196, 2012.

[15] C. Neti, G. Potamianos, J. Luettin, I. Matthews, H. Glotin, D. Vergyri, J. Sison, A. Mashari, J. Zhou, Audio-Visual Speech Recognition, Workshop 2000 Final Report, 2000.

[16] Fei Sha, Lawrence K. Saul, Analysis and Extension of Spectral Methods for Nonlinear Dimensionality Reduction, International Conference on Machine Learning, Bonn 2005.

[17] Lawrence Saul, Kilian Weinberger, Fei Sha, Jihun Ham, and Daniel Lee, Spectral Methods for Dimensionality Reduction, in "Semi-supervised Learning", MIT, 2006

[18] Junping Zhang, Stan Li, and Jue Wang, Manifold Learning and Application in Recognition, Intelligent Multimedia Processing with Soft Computing, Studies in Fuzziness and Soft Computing, Vol.168, pp.281-300, Springer Ed., 2005.

[19] R. Seymour, D. Stewart, and j. Ming, Comparasion of Image Transform-Based Features for Visual Speech Recognition in Clean and Corrupted Videos, EURASIP Journal on Image and Video Processing, 2008.

[20] T. Pao, W. Liao, C. Lin, Automatic Visual Feature Extraction for Mandarin Ausio-Visual Speech Recognition, International Conference on Systems, Man, and Cybernetics, 2009.

[21] T. Drugman, M. Gurban, and J.-F. Thiran, Relevant Feature Selection for Audio-Visual Speech Recognition, IEEE Workshop on Multimedia Signal Processing, 2007.

[22] G. Galatas, G. Potamianos, A. Papangellis, F. Makedon, Audio Visual Speech Recognition in Noisy Visual Environments, PETRA'11, Crete 2011.

[23] V. Pera, The Language Model of the ABCP1 System, Tech. Rep., FEUP, Porto, 2011.

[24] V. Pera, An Overview of the ABCP1 System, Tech. Rep., FEUP Porto, 2012.

[25] Naotoshi Seo, Tutorial: OpenCV haartraining, http://note.sonots.com/SciSoftware/haartraining.html.

[26] Blaise Bairney, Lawrence Livermore National Laboratory Tutorial: OpenMP, https://computing.llnl.gov/tutorials/openMP/.

A The SVAAM code

In the Web address `http://speech-rec-vcp.com` is availabe the file `lrn2m_wr-1.0.cpp`, with the code shown in this appendix, and also the header files `Lrn2m_wr.h` and `CompFlt.h`.

```
/* ------------------------------------------------------------------ */
/*                                                                    */
/*                      -   --  --                                    */
/*                    /\   |-) |   |--)  AUDIO-VISUAL                 */
/*                   /--\  |-) |-- |     SPEECH RECOGNITION           */
/*                  ======================  SOFTWARE                  */
/*                                                                    */
/*                                                                    */
/* ------------------------------------------------------------------ */
/* developed by:                                                      */
/*      Vitor Manuel Martins Cicouro de Pera                          */
/*      http://speech-rec-vcp.com                                     */
/*      FEUP (Porto - Portugal)                  25/Feb/2012          */
/*                                                                    */
/* credits also due to:                                               */
/*      The OpenCV library                                            */
/*      http://opencv.org                                             */
/*                                                                    */
/*                 ---------------------------                        */
/*                                                                    */
/*   This software is free for research use, respecting also          */
/*   the terms under the (OpenCV) BSD license.                        */
/*    ** See the file License for the Conditions of Use  **           */
/*    **      This banner notice must not be removed      **         */
/* ------------------------------------------------------------------ */
/*      File: lrn2m_wr-1.0.cpp       (0.9.2)                          */
/*                                                                    */
/*      ROI detection and tracking plus features extraction           */
/*      for Audio Visual Speech Recognition application               */
/*                                                                    */
/* ------------------------------------------------------------------ */



#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <fstream>
#include <vector>
#include <stdio.h>
#include <math.h>

#include "Lrn2m_wr.h"
#include "CompFlt.h"

```

```
49
50 //-------------------------------------------------------------------------------
51 int main(int ac, char** av)
52 {
53   if( ac != 2 )
54   { help();
55     return -1; }
56   initLrn2m();
57
58   capture = openInputVideo( av[1] );
59   preserve = openOutputVideo( capture );
60
61   CascadeClassifier cascadeFace = loadCascadeClassifier( cascadeFaceName );
62   CascadeClassifier cascadeNose = loadCascadeClassifier( cascadeNoseName );
63   CascadeClassifier cascadeEyeL = loadCascadeClassifier( cascadeEyeLName );
64   CascadeClassifier cascadeEyeR = loadCascadeClassifier( cascadeEyeRName );
65
66   cvNamedWindow( "video", 1 );
67
68   int x1 (0), xNose (0), xEyeL (0), xEyeR (0),
69       y1 (0), yNose (0), yEyeL (0), yEyeR (0),
70       w1 (0), wNose (0), wEyeL (0), wEyeR (0),
71       h1 (0), hNose (0), hEyeL (0), hEyeR (0);
72
73   if( faceDetection( cascadeFace, x1, y1, w1, h1 ) == -1 ) return -1;
74
75   Mat frame, ft0, ft;
76   Point noseC, eyeLC, eyeRC;
77
78   capture >> frame;
79   faceFeatureDetection( frame, cascadeNose, scaleNose, x1, y1, w1, h1, xNose
       , yNose, wNose, hNose, noseC, 1 );
80   faceFeatureDetection( frame, cascadeEyeL, scaleEyeL, x1, y1, w1, h1, xEyeL
       , yEyeL, wEyeL, hEyeL, eyeLC, 2 );
81   faceFeatureDetection( frame, cascadeEyeR, scaleEyeR, x1, y1, w1, h1, xEyeR
       , yEyeR, wEyeR, hEyeR, eyeRC, 3 );
82   roiDetection( frame, noseC, eyeLC, eyeRC );
83   preserve << frame;
84
85   cv::imshow( "video", frame );
86   if(waitKey(30) >= 0) return -1;
87
88   cout << "\nstart tracking..." << endl;
89 #ifdef _SHOW_TIME
90   printf( "ttime (ms)\n");
91   printf( "nose\teyeL\teyeR\troi1\troi2\tALL\n");
92   printf( "-------------------------------------------------\n");
93 #endif
94   for(;;)
95   {
96     capture >> frame;
97
98 #ifdef _SHOW_TIME
99     double t = (double)cvGetTickCount();
```

```
100 #endif
101
102     noseTracking( frame, cascadeNose, xNose, yNose, wNose, hNose, noseC );
103
104     eyeLTracking( frame, cascadeEyeL, xEyeL, yEyeL, wEyeL, hEyeL, eyeLC );
105
106     eyeRTracking( frame, cascadeEyeR, xEyeR, yEyeR, wEyeR, hEyeR, eyeRC );
107
108     if( roiLocalization( frame, noseC, eyeLC, eyeRC, ft0 ) == -1 ) return
        -1;
109
110     roiTuning( ft0, ft );
111 //
112 //  DCT_FEATURES := DCT-TRANSFORM ( FT )        - openCV function
113 //  GEO_FEATURES := GET_GEO_FEATURES ( ... )    - read from pre-normalized
        ROI rectangle
114 //  SAVE DCT_FEATURES & GEO_FEATURES
115 //
116     if( showImages( frame, ft ) == -1 ) return -1;
117
118 #ifdef _SHOW_TIME
119     t = (double)cvGetTickCount() - t;
120     printf( "%.4g\n", t/((double)cvGetTickFrequency()*1000.) );
121 #endif
122
123     preserve << frame;
124   }
125   cvDestroyWindow("video");
126   return 0;
127 }
128
129
130 //-------------------------------------------------------------------------
131 void help()
132 {
133   cerr << "\nERROR: Wrong number of parameters!\n" << endl;
134   cout << "Usage:\n\n"
135           "./lrn2m_wr-1.0 <video-filename>\n\n" << endl;
136 //  waitKey(100);
137 }
138
139
140 //-------------------------------------------------------------------------
141 void initLrn2m()
142 {
143 #ifdef _TM1
144   Mat templ = imread( "template.jpg", 1 );
145   cvtColor( templ, tm_templ, CV_BGR2GRAY );
146 #endif
147
148   for( int i=0; i<maxHalfOf; i++)
149   { halfOf[i] = cvRound(i*0.5);
150   }
151   for( int i=0; i<maxOnePlusK0LOf; i++)
```

```
152   { onePlusK0LOf[i] = cvRound(i*(1.0+k0L));
153   }
154   for( int i=0; i<maxK0LOf; i++)
155   { K0LOf[i] = cvRound(i*k0L);
156   }
157   for( int i=0; i<maxK0WOf; i++)
158   { K0WOf[i] = cvRound(i*k0W);
159   }
160   for( int i=0; i<maxK0HOf; i++)
161   { K0HOf[i] = cvRound(i*k0H);
162   }
163   //
164   cout << "\nThis program extracts visual-speech cues from the mouth region
        (ROI), to use in an audio-visual speech recognizer (AVSR).\n"
165         "Boosted Cascade Classifiers are used to detect and track the nose
        and the eyes, auxiliary face features, based on Haar features.\n"
166      "Then a Template Matching algorithm is used to locate precisely the
        mouth region.\n"
167         "Using OpenCV version " << CV_VERSION << "\n\n"
168         "Press any key to abort execution\n\n" << endl;
169   waitKey(1000);
170 }
171
172
173 //--------------------------------------------------------------------------
174 VideoCapture openInputVideo( string videoFile )
175 {
176   VideoCapture capture(videoFile);
177   if (!capture.isOpened())
178   { cerr << "ERROR: Failed to open the video file!\n" << endl;
179 //    return -1;
180   }
181   return capture;
182 }
183
184
185 //--------------------------------------------------------------------------
186 VideoWriter openOutputVideo( VideoCapture capture )
187 {
188   VideoWriter preserve;
189 //  int ex = static_cast<int>(capture.get(CV_CAP_PROP_FOURCC));  // same
        codec type than input file
190   Size preserveS = Size((int) capture.get(CV_CAP_PROP_FRAME_WIDTH), (int)
        capture.get(CV_CAP_PROP_FRAME_HEIGHT));
191 //  preserve.open("result.avi", ex, capture.get(CV_CAP_PROP_FPS), preserveS,
        true);
192   preserve.open("result.avi", CV_FOURCC('M','J','P','G'), capture.get(
        CV_CAP_PROP_FPS), preserveS, true);
193   if ( !preserve.isOpened()){
194     cout << "couldn't open writer" << endl;
195 //    return 1;
196   }
197   return preserve;
198 }
```

```
199
200
201 //-------------------------------------------------------------------------------
202 CascadeClassifier loadCascadeClassifier( string name )
203 {
204   CascadeClassifier cascade;
205   if( !cascade.load( name ) )
206   { cerr << "ERROR: Could not load " << name << " classifier!" << endl;
207 //    return -1;
208   }
209   return cascade;
210 }
211
212
213 //-------------------------------------------------------------------------------
214 int faceDetection( CascadeClassifier& cascadeFace, int& x1, int& y1, int& w1
      , int& h1 )
215 {
216   cout << "\n\nface detection..." << endl;
217   Mat frame;
218   //----- tmp -----
219   for (int i=0;i<200;i++)
220     capture >> frame;
221   //---------------
222   detect1( frame, cascadeFace, scaleFace, x1, y1, w1, h1 );
223
224   cv::imshow( "video", frame );
225   if(waitKey(30) >= 0) return -1;
226
227   int x, y;
228   int i=1;
229   while( i < initialFrames )
230   { x = x1;
231     y = y1;
232     capture >> frame;
233     detect1( frame, cascadeFace, scaleFace, x1, y1, w1, h1 );
234     if( (x1-x)*(x1-x)+(y1-y)*(y1-y) > initialDrift ) i = 0;
235     i++;
236
237     cv::imshow( "video", frame );
238     if(waitKey(30) >= 0) return -1;
239
240     preserve << frame;
241   }
242   return 0;
243 }
244
245
246 //-------------------------------------------------------------------------------
247 void faceFeatureDetection( Mat& frame, CascadeClassifier& cascade, int scale
      , int x1, int y1, int w1, int h1, int& x2, int& y2, int& w2, int& h2,
      Point& fC, int f )
248 {
249   int x (0), y (0),
```

```
250       xt, yt, wt, ht;
251   Mat ft;
252
253   capture >> frame;
254   switch( f ) {
255     case 1:
256     cout << "\nnose detection..." << endl;
257     xt = cvRound(w1*xMinFNose);
258     yt = cvRound(h1*yMinFNose);
259     wt = cvRound(w1*xWFNose);
260     ht = cvRound(h1*yHFNose);
261     break;
262     case 2:
263     cout << "\nleft-eye detection..." << endl;
264     xt = cvRound(w1*xMinFEyeL);
265     yt = cvRound(h1*yMinFEyeL);
266     wt = cvRound(w1*xWFEyeL);
267     ht = cvRound(h1*yHFEyeL);
268     break;
269     case 3:
270     cout << "\nright-eye detection..." << endl;
271     xt = cvRound(w1*xMinFEyeR);
272     yt = cvRound(h1*yMinFEyeR);
273     wt = cvRound(w1*xWFEyeR);
274     ht = cvRound(h1*yHFEyeR);
275   }
276   ft = frame(Rect(x1+xt,y1+yt,wt,ht));
277   detect1( ft, cascade, scale, x, y, w2, h2 );
278   x2 = x1 + xt + x - 1;
279   y2 = y1 + yt + y - 1;
280   fC.x = x2 + cvRound(w2*0.5);
281   fC.y = y2 + cvRound(h2*0.5);
282 }
283
284
285 //-------------------------------------------------------------------------
286 // ********************************************************* change/tune
287 int roiDetection( Mat& frame, Point& noseC, Point& eyeLC, Point& eyeRC )
288 {
289   int wr, hr;
290   Point roi1, roi2;
291
292   //-------------------------------------------------------------------| tn
293   wr = cvRound( k0W * ( eyeLC.x - eyeRC.x ) );
294   hr = cvRound( k0H * ( noseC.y - ( eyeRC.y + eyeLC.y ) * 0.5 ) );
295   roi1.x = noseC.x - cvRound( wr * ( 0.5 - k0S ) );
296   roi1.y = noseC.y + cvRound( k0L * ( noseC.y - ( eyeRC.y + eyeLC.y ) * 0.5
        ) );
297   roi2.x = roi1.x + cvRound( wr * ( 1 + k0S ) );
298   roi2.y = roi1.y + hr;
299 //  rectangle( frame, roi1, roi2, Scalar(255,255,255), 2, 8, 0 );
300   //----- tshow -----
301 /*
302   Mat ft = frame( Rect( roi1.x, roi1.y, wr, hr ) );
```

```
303   cv::imshow( "roi", ft );
304   if( waitKey(30) >= 0 ) return -1;
305 */
306   //-----------------
307
308 #ifndef _RI_BW
309   // FLT ------------------------------------------------
310   CompFlt compFlt1;  // w/o getData (w/ in v.1.0+)
311   compFlt1.getPar( fltPrm );
312   fltPrm[0] *= fltMp;
313   fltPrm[1] *= fltMp;
314   fltPrm[2] *= fltMp;
315   cout << "\nB = " << fltPrm[0] << "\tG = " << fltPrm[1] << "\tR = " <<
        fltPrm[2] << endl;
316 #endif
317
318   return 0;
319 }
320
321
322 //-------------------------------------------------------------------------
323 void noseTracking( Mat& frame, CascadeClassifier& cascade, int& x2, int& y2,
        int& w2, int& h2, Point& fC )
324 {
325 #ifdef _SHOW_TIME
326   double t1 = (double)cvGetTickCount();
327 #endif
328   int x3 = x2-deltaNose;
329   int y3 = y2-deltaNose;
330   int w3 = w2+2*deltaNose;
331   int h3 = h2+2*deltaNose;
332   Mat img = frame(Rect(x3,y3,w3,h3));
333
334   Mat gray, smallImg( cvRound (img.rows/scaleNose), cvRound(img.cols/
        scaleNose), CV_8UC1 );
335   cvtColor( img, gray, CV_BGR2GRAY );
336   resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
337   equalizeHist( smallImg, smallImg );
338
339   vector<Rect> faces;
340   cascade.detectMultiScale( smallImg, faces,
341     1.15, // 1.1
342     2,    // 5
343     0
344     //|CV_HAAR_FIND_BIGGEST_OBJECT,
345     //|CV_HAAR_DO_ROUGH_SEARCH,
346     |CV_HAAR_SCALE_IMAGE,      // <----
347     Size(20, 20) );  // 20, 20
348
349   int xpt, ypt, w2t, h2t, xp, yp;
350   int xmin = deltaNose,
351       xmax = img.cols - deltaNose,
352       ymin = deltaNose,
353       ymax = img.rows - deltaNose;
```

```
354   int hdm, hdx, hdy, hdt;
355   hdm = 10000;
356   for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end(); r++
        )
357   {
358     xpt = cvRound(r->x*scaleNose);
359     ypt = cvRound(r->y*scaleNose);
360     w2t = cvRound(r->width*scaleNose);
361     h2t = cvRound(r->height*scaleNose);
362     hdx = (xmin-xpt)*(xmin-xpt);
363     if ( (hdt = (xmax-xpt-w2t)*(xmax-xpt-w2t)) > hdx )
364       hdx = hdt;
365     hdy = (ymin-ypt)*(ymin-ypt);
366     if ( (hdt = (ymax-ypt-h2t)*(ymax-ypt-h2t)) > hdy )
367       hdy = hdt;
368     if ( hdx + hdy < hdm )
369     { hdm = hdx + hdy;
370       xp = xpt;
371       yp = ypt;
372       w2 = w2t;
373       h2 = h2t;
374     }
375   }
376   x2 = x3 + xp - 1;
377   y2 = y3 + yp - 1;
378   fC.x = x2 + halfOf[w2];
379   fC.y = y2 + halfOf[h2];
380
381 //   Point pa, pb;
382 //   pa.x = xp;
383 //   pa.y = yp;
384 //   pb.x = xp+w2;
385 //   pb.y = yp+h2;
386 //   rectangle( img, pa, pb, CV_RGB(255,255,255), 1, 8, 0 );
387   circle( frame, fC, 3, Scalar(255,255,255), 3, 8, 0 );
388 #ifdef _SHOW_TIME
389   t1 = (double)cvGetTickCount() - t1;
390   printf( "%.4g\t", t1/((double)cvGetTickFrequency()*1000.) );
391 #endif
392 }
393
394
395 //-------------------------------------------------------------------------
396 void eyeLTracking( Mat& frame, CascadeClassifier& cascade, int& x2, int& y2,
        int& w2, int& h2, Point& fC )
397 {
398 #ifdef _SHOW_TIME
399   double t1 = (double)cvGetTickCount();
400 #endif
401   int x3 = x2-deltaEyeL;
402   int y3 = y2-deltaEyeL;
403   int w3 = w2+2*deltaEyeL;
404   int h3 = h2+2*deltaEyeL;
405   Mat img = frame(Rect(x3,y3,w3,h3));
```

```
406
407   Mat gray, smallImg( cvRound (img.rows/scaleEyeL), cvRound(img.cols/
        scaleEyeL), CV_8UC1 );
408   cvtColor( img, gray, CV_BGR2GRAY );
409   resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
410   equalizeHist( smallImg, smallImg );
411
412   vector<Rect> faces;
413   cascade.detectMultiScale( smallImg, faces,
414     1.25, // 1.1
415     2,    // 5
416     0
417     //|CV_HAAR_FIND_BIGGEST_OBJECT,
418     //|CV_HAAR_DO_ROUGH_SEARCH,
419     |CV_HAAR_SCALE_IMAGE,      // <----
420     Size(20, 20) );  // 20, 20
421
422   int xpt, ypt, w2t, h2t, xp, yp;
423   int xmin = deltaEyeL,
424       xmax = img.cols - deltaEyeL,
425       ymin = deltaEyeL,
426       ymax = img.rows - deltaEyeL;
427   int hdm, hdx, hdy, hdt;
428   hdm = 10000;
429   for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end(); r++
        )
430   {
431     xpt = cvRound(r->x*scaleEyeL);
432     ypt = cvRound(r->y*scaleEyeL);
433     w2t = cvRound(r->width*scaleEyeL);
434     h2t = cvRound(r->height*scaleEyeL);
435     hdx = (xmin-xpt)*(xmin-xpt);
436     if ( (hdt = (xmax-xpt-w2t)*(xmax-xpt-w2t)) > hdx )
437       hdx = hdt;
438     hdy = (ymin-ypt)*(ymin-ypt);
439     if ( (hdt = (ymax-ypt-h2t)*(ymax-ypt-h2t)) > hdy )
440       hdy = hdt;
441     if ( hdx + hdy < hdm )
442     { hdm = hdx + hdy;
443       xp = xpt;
444       yp = ypt;
445       w2 = w2t;
446       h2 = h2t;
447     }
448   }
449   x2 = x3 + xp - 1;
450   y2 = y3 + yp - 1;
451   fC.x = x2 + halfOf[w2];
452   fC.y = y2 + halfOf[h2];
453
454 //   Point pa, pb;
455 //   pa.x = xp;
456 //   pa.y = yp;
457 //   pb.x = xp+w2;
```

```
458 //  pb.y = yp+h2;
459 //  rectangle( img, pa, pb, CV_RGB(255,255,255), 1, 8, 0 );
460   circle( frame, fC, 3, Scalar(255,255,255), 3, 8, 0 );
461 #ifdef _SHOW_TIME
462   t1 = (double)cvGetTickCount() - t1;
463   printf( "%.4g\t", t1/((double)cvGetTickFrequency()*1000.) );
464 #endif
465 }
466
467
468 //-------------------------------------------------------------------------------
469 void eyeRTracking( Mat& frame, CascadeClassifier& cascade, int& x2, int& y2,
        int& w2, int& h2, Point& fC )
470 {
471 #ifdef _SHOW_TIME
472   double t1 = (double)cvGetTickCount();
473 #endif
474   int x3 = x2-deltaEyeR;
475   int y3 = y2-deltaEyeR;
476   int w3 = w2+2*deltaEyeR;
477   int h3 = h2+2*deltaEyeR;
478   Mat img = frame(Rect(x3,y3,w3,h3));
479
480   Mat gray, smallImg( cvRound (img.rows/scaleEyeR), cvRound(img.cols/
        scaleEyeR), CV_8UC1 );
481   cvtColor( img, gray, CV_BGR2GRAY );
482   resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
483   equalizeHist( smallImg, smallImg );
484
485   vector<Rect> faces;
486   cascade.detectMultiScale( smallImg, faces,
487     1.25, // 1.1
488     2,    // 5
489     0
490     //|CV_HAAR_FIND_BIGGEST_OBJECT,
491     //|CV_HAAR_DO_ROUGH_SEARCH,
492     |CV_HAAR_SCALE_IMAGE,      // <----
493     Size(20, 20) );  // 20, 20
494
495   int xpt, ypt, w2t, h2t, xp, yp;
496   int xmin = deltaEyeR,
497       xmax = img.cols - deltaEyeR,
498       ymin = deltaEyeR,
499       ymax = img.rows - deltaEyeR;
500   int hdm, hdx, hdy, hdt;
501   hdm = 10000;
502   for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end(); r++
        )
503   {
504     xpt = cvRound(r->x*scaleEyeR);
505     ypt = cvRound(r->y*scaleEyeR);
506     w2t = cvRound(r->width*scaleEyeR);
507     h2t = cvRound(r->height*scaleEyeR);
508     hdx = (xmin-xpt)*(xmin-xpt);
```

```
509     if ( (hdt = (xmax-xpt-w2t)*(xmax-xpt-w2t)) > hdx )
510       hdx = hdt;
511     hdy = (ymin-ypt)*(ymin-ypt);
512     if ( (hdt = (ymax-ypt-h2t)*(ymax-ypt-h2t)) > hdy )
513       hdy = hdt;
514     if ( hdx + hdy < hdm )
515     { hdm = hdx + hdy;
516       xp = xpt;
517       yp = ypt;
518       w2 = w2t;
519       h2 = h2t;
520     }
521   }
522   x2 = x3 + xp - 1;
523   y2 = y3 + yp - 1;
524   fC.x = x2 + halfOf[w2];
525   fC.y = y2 + halfOf[h2];
526
527 //  Point pa, pb;
528 //  pa.x = xp;
529 //  pa.y = yp;
530 //  pb.x = xp+w2;
531 //  pb.y = yp+h2;
532 //  rectangle( img, pa, pb, CV_RGB(255,255,255), 1, 8, 0 );
533   circle( frame, fC, 3, Scalar(255,255,255), 3, 8, 0 );
534 #ifdef _SHOW_TIME
535   t1 = (double)cvGetTickCount() - t1;
536   printf( "%.4g\t", t1/((double)cvGetTickFrequency()*1000.) );
537 #endif
538 }
539
540
541 //-------------------------------------------------------------------------------------------
542 int roiLocalization( Mat& frame, Point& noC, Point& elC, Point& erC, Mat&
      ft0 )
543 {
544 #ifdef _SHOW_TIME
545   double t1 = (double)cvGetTickCount();
546 #endif
547   int wr, hr, wrG, hrG;
548   Point oPt, pPt;
549
550   double theta = atan( (double)(elC.y-erC.y) / (double)(elC.x-erC.x) );
551   if( theta > -thetaThr && theta < thetaThr ) // small angles
552   {
553     roiRectColor = CV_RGB(255,255,255);
554 #ifdef SMALL_ANGLE_VSMPL
555     oPt.x = halfOf[ erC.x + elC.x ];
556     oPt.y = halfOf[ erC.y + elC.y ];
557     int dex = elC.x - erC.x;
558     int dny = noC.y - oPt.y;
559     pPt.x = cvRound( keL * oPt.x + kmL * noC.x + k0S * dex );
```

```
560     pPt.y = cvRound( noC.y + k0L * dny );
561     wr = k0W * dex;
562     hr = k0H * dny;
563 #else
564 // ********************************************************************* xx
565     int dex = elC.x - erC.x;
566     int dey = elC.y - erC.y;
567     double ded = sqrt( dex * dex + dey * dey );
568     double ux = dex / ded;
569     double uy = dey / ded;
570     double psl = ux * (noC.x-erC.x) + uy * (noC.y-erC.y);
571     oPt.x = erC.x + cvRound( psl * ux );
572     oPt.y = erC.y + cvRound( psl * uy );
573 //    oPt.x = erC.x + cvRound( psl * ux + k0S*(elC.x-erC.x) );
574 //    oPt.x = cvRound( 0.5 * ( erC.x + elC.x ) );
575 //    oPt.y = cvRound( 0.5 * ( erC.y + elC.y ) );
576     pPt.x = oPt.x;
577     pPt.y = oPt.y + onePlusK0LOf[ noC.y-oPt.y ];
578 //    pPt.x = noC.x;
579 //    pPt.y = noC.y + cvRound( k0L * (noC.y-oPt.y) );
580 //    pPt.y = oPt.y + cvRound( (1.0+k0L) * (noC.y-oPt.y) );
581     wr = K0WOf[ elC.x-erC.x ];
582     hr = K0HOf[ noC.y-oPt.y ];
583 //    wr = cvRound( k0W * ( elC.x - erC.x ) );
584 //    hr = cvRound( k0H * ( noC.y - oPt.y ) );
585 #endif   // SMALL_ANGLE_VSMPL
586     Rect roi_rect( pPt.x-wr, pPt.y-hr, 2*wr, 2*hr );
587     ft0 = frame( Rect( roi_rect ) );
588   }
589   else   // large angles
590   {
591     roiRectColor = CV_RGB(255,0,0);
592 #ifdef LARGE_ANGLE_VSMPL
593 // ********************************************************************* xx
594     oPt.x = halfOf[ erC.x + elC.x ];
595     oPt.y = halfOf[ erC.y + elC.y ];
596     int dex = elC.x - erC.x;
597     int dey = elC.y - erC.y;
598     float ux = dex / sqrt( dex * dex + dey * dey );
599     float uy = dey / sqrt( dex * dex + dey * dey );
600     double psl = ux * ( noC.x - erC.x ) + uy * ( noC.y - erC.y );
601     Point olPt;
602     olPt.x = erC.x + cvRound( psl * ux );
603     olPt.y = erC.y + cvRound( psl * uy );
604     Point dto;
605     dto.x = keL * oPt.x + (kmL-1) * olPt.x; //+ k0S * dex;
606     dto.y = keL * oPt.y + (kmL-1) * olPt.y;
607     Point dno;
608     dno.x = noC.x - olPt.x;
609     dno.y = noC.y - olPt.y;
610     pPt.x = noC.x + cvRound( k0L * dno.x + dto.x );
611     pPt.y = noC.y + cvRound( k0L * dno.y + dto.y );
612     wr = cvRound( k0W * sqrt( dex * dex + dey * dey ) );
613     hr = cvRound( k0H * sqrt( dno.x * dno.x + dno.y * dno.y ) );
```

```
614     if( theta > 0 )
615     { wrG = cvRound( (double)wr * cos(theta) + (double)hr * sin(theta) );
616       hrG = cvRound( (double)hr * cos(theta) + (double)wr * sin(theta) );
617     } else
618     { wrG = cvRound( (double)wr * cos(theta) - (double)hr * sin(theta) );
619       hrG = cvRound( (double)hr * cos(theta) - (double)wr * sin(theta) );
620     }
621 #else
622 // ************************************************************** xx
623     double ux = (elC.x-erC.x) / sqrt( (elC.x-erC.x) * (elC.x-erC.x) + (elC.y
    -erC.y) * (elC.y-erC.y) );
624     double uy = (elC.y-erC.y) / sqrt( (elC.x-erC.x) * (elC.x-erC.x) + (elC.y
    -erC.y) * (elC.y-erC.y) );
625     double psl = ux * (noC.x-erC.x) + uy * (noC.y-erC.y);
626     oPt.x = erC.x + cvRound( psl*ux );
627     oPt.y = erC.y + cvRound( psl*uy );
628 //    oPt.x = cvRound(0.5*(erC.x+elC.x));
629 //    oPt.y = cvRound(0.5*(erC.y+elC.y));
630     pPt.x = cvRound( oPt.x + (1.0+k0L) * (noC.x-oPt.x) );
631 //    pPt.x = noC.x+cvRound(k0L*(noC.x-oPt.x));
632     pPt.y = oPt.y + cvRound( (1.0+k0L) * (noC.y-oPt.y) );
633 //    pPt.y = noC.y+cvRound(k0L*(noC.y-oPt.y));
634     wr = cvRound( k0W * sqrt( (elC.x-erC.x) * (elC.x-erC.x) + (elC.y-erC.y)
    * (elC.y-erC.y) ) );
635     hr = cvRound( k0H * sqrt( (noC.x-oPt.x) * (noC.x-oPt.x) + (noC.y-oPt.y)
    * (noC.y-oPt.y) ) );
636     if( theta > 0 )
637     { wrG = cvRound( (double)wr * cos(theta) + (double)hr * sin(theta) );
638       hrG = cvRound( (double)hr * cos(theta) + (double)wr * sin(theta) );
639     }
640     else
641     { wrG = cvRound( (double)wr * cos(theta) - (double)hr * sin(theta) );
642       hrG = cvRound( (double)hr * cos(theta) - (double)wr * sin(theta) );
643     }
644 #endif   // LARGE_ANGLE_VSMPL
645     Mat dst_img = frame.clone();
646     // set ROI
647     Rect roi_rect( pPt.x - wrG, pPt.y - hrG, 2 * wrG, 2 * hrG );
648     Mat src_roi( frame, roi_rect );
649     Mat dst_roi( dst_img, roi_rect );
650     // with specified parameters (angle, rotation center, scale)
651     //calculate affine T matrix
652     double angle = theta*Rad2Grad,
653      scale = 1.0;
654     Point2d center( src_roi.cols*0.5, src_roi.rows*0.5 );
655     const Mat affine_matrix = getRotationMatrix2D( center, angle, scale );
656     // rotate the image by warpAffine taking the affine matrix
657     warpAffine( src_roi, dst_roi, affine_matrix, dst_roi.size(),
    INTER_LINEAR, BORDER_CONSTANT, Scalar::all(255) );
658     ft0 = dst_roi( Rect( center.x - wr, center.y - hr, 2 * wr, 2 * hr ) );
659   }
660   /*
661   // Show broad ROI
```

```
662   rectangle( frame, Point( pPt.x-wr, pPt.y-hr ), Point( pPt.x+wr, pPt.y+hr )
        ,\
663     Scalar::all(0), 1, 8, 0 );
664   */
665   if( pPt.y+hr >= frame.rows-faceTooClose )
666   { cout << endl << "FACE TOO CLOSE! STOP." << endl;
667     return -1; }
668 #ifdef _SHOW_TIME
669   t1 = (double)cvGetTickCount() - t1;
670   printf( "%.1g\t", t1/((double)cvGetTickFrequency()*1000.) );
671 #endif
672   return 0;
673 }
674
675
676 //-------------------------------------------------|change ~_RI_BW and ~_TM1
677 void roiTuning( Mat& ft0, Mat& fltt )
678 {
679 #ifdef _SHOW_TIME
680   double t1 = (double)cvGetTickCount();
681 #endif
682
683 #ifdef _RI_BW    // standard (not (tmp) BGR2FltBW-v.0)
684
685   Mat flt;
686   cvtColor( ft0, flt, CV_BGR2GRAY );
687
688 #ifdef _TM1    // Template Matching (1)
689
690   resize( flt, flt, Size(WrN, HrN), 0, 0, CV_INTER_LINEAR );    // v.0
691   int  tm_x = cvRound(flt.cols*TmX),
692        tm_y = TmY,
693        tm_w = cvRound(flt.cols*TmW),
694        tm_h = cvRound(flt.cols*TmH);
695   Mat flts = flt( Rect( tm_x, tm_y, tm_w, tm_h ) );
696   //char* method[6] = { "SQDIFF (0)",   "SQDIFF NORMED (1)",
697   //                    "TM CCORR (2)", "TM CCORR NORMED (3)",
698   //                    "TM COEFF (4)", "TM COEFF NORMED (5)" };
699   int match_method = 5;    // 3
700   /// Create the result matrix
701   Mat result;
702   int result_cols = flts.cols - tm_templ.cols + 1;
703   int result_rows = flts.rows - tm_templ.rows + 1;
704   result.create( result_cols, result_rows, CV_32FC1 );
705   /// Do the Matching and Normalize
706   matchTemplate( flts, tm_templ, result, match_method );
707   normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );
708   /// Localizing the best match with minMaxLoc
709   double minVal, maxVal;
710   Point minLoc, maxLoc, matchLoc;
711   minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );
712   // For SQDIFF and SQDIFF_NORMED, the best matches are lower values.
713   // For all the other methods, the higher the better
714   matchLoc = maxLoc;    // minLoc
```

```
715 /*
716 // Show detected region
717   Point matchLocShifted;
718   matchLocShifted.x = matchLoc.x + tm_x;
719   matchLocShifted.y = matchLoc.y + tm_y;
720   rectangle( flt, matchLocShifted, Point( matchLocShifted.x + tm_templ.cols
        ,\
721                 matchLocShifted.y + tm_templ.rows ), Scalar::all(0), 1, 8,
        0 );
722 */
723   Point match_1,
724   match_2;
725   //-------------------------------------------------------------------| ADAPT
726   match_1.x = matchLoc.x + tm_x - RBleft;
727   match_1.y = matchLoc.y + tm_y - RBup;
728   match_2.x = matchLoc.x + tm_x + tm_templ.cols + RBright;
729   match_2.y = matchLoc.y + tm_y + tm_templ.rows + RBdown;
730   //
731   if( match_2.x > flt.cols) match_2.x = flt.cols;    // w/o adapt
732   if( match_2.y > flt.rows) match_2.y = flt.rows;    // w/o adapt
733
734   fltt = flt( Rect( match_1.x, match_1.y, match_2.x-match_1.x, match_2.y-
        match_1.y ) );
735   GaussianBlur( fltt, fltt, Size(3,3), 0, 0);
736   resize( fltt, fltt, Size(32, 32), 0, 0, CV_INTER_LINEAR );
737
738 #else // no Template Matching (1)
739
740   //-------------------------------------------------------------------| CHANGE
741
742   GaussianBlur( flt, flt, Size(3,3), 0, 0);
743 //  Mat frot;
744 //  GaussianBlur( flt, frot, Size(5,5), 1.5, 1.5);
745 //  Canny(frot, frot, 25, 50, 3);
746   fltt = flt.clone();
747 //  Mat fltt = flt.clone();
748
749   Mat grad_y;
750
751   // clip-y
752   int scale = 1;
753   int delta = 20;
754   Sobel( flt, grad_y, CV_8U, 0, 1, 5, scale, delta, BORDER_DEFAULT );
755   convertScaleAbs( grad_y, flt );
756   int xi = halfOf[ flt.cols ] - xdDL;
757   int xf = halfOf[ flt.cols ] + xdDL;
758   int y0, y1, pv1 = yThr;
759   for( int y = yBegin; y < flt.rows; y++ )
760   { int pv2 = 0;
761     for( int x = xi; x < xf; x++ )
762     { pv2 += flt.at<uchar>(y+1,x);
763     }
764     if( pv1 < yThr && pv2 < yThr )
765     { y0 = y + yShift;
```

```
766       y1 = y0 + cvRound( yHrF * hr );
767       break;
768     }
769     pv1 = pv2;
770   }
771   fltt = fltt( Rect( 0, y0, flt.cols, y1-y0-1 ) );
772
773   // clip-x (v.1.0 symm)
774 //  blur( fltt, fltt, Size(5,5), Point(-1,-1));
775 //  GaussianBlur( fltt, fltt, Size(7,7), 0, 0);
776   scale = 1;
777   delta = 20;
778   Mat grad_x;
779   Sobel( fltt, grad_x, CV_8U, 0, 1, 5, scale, delta, BORDER_DEFAULT );
780 //  Scharr( flt, grad_y, CV_8U, 0, 1, scale, delta, BORDER_DEFAULT );
781   convertScaleAbs( grad_x, fltt );
782
783   int xLong = cvRound( xLongF * fltt.cols  );
784   int xDlt = cvRound( xDltF * fltt.cols  );
785   int xLg = cvRound( xLgF * fltt.cols  );
786   int yUp = cvRound( yUpF * fltt.rows  );
787   int yDown = cvRound( yDownF * fltt.rows  );
788
789   int xc = cvRound( fltt.cols * 0.5 );
790   int x;
791   int v = 0;
792   for( int y = yUp; y < yDown; y++ )
793   { for( int i = 1; i <= xLg; i++ )
794 //    { v += (fltt.at<uchar>(y,xc+xDlt+i) - fltt.at<uchar>(y,xc-xDlt-i)) *
795 //    (fltt.at<uchar>(y,xc+xDlt+i) - fltt.at<uchar>(y,xc-xDlt-i));
796     { if( fltt.at<uchar>(y,xc+xDlt+i) > fltt.at<uchar>(y,xc-xDlt-i) )
797       { v += fltt.at<uchar>(y,xc+xDlt+i) - fltt.at<uchar>(y,xc-xDlt-i);
798       } else
799       { v -= fltt.at<uchar>(y,xc+xDlt+i) - fltt.at<uchar>(y,xc-xDlt-i);
800       }
801     }
802   }
803   for( x = xc+1; x < xc+xLong; x++ )
804   { int u = 0;
805     for( int y = yUp; y < yDown; y++ )
806     { for( int i = 1; i <= xLg; i++ )
807 //      { u += (fltt.at<uchar>(y,x+xDlt+i) - fltt.at<uchar>(y,x-xDlt-i)) *
808 //      (fltt.at<uchar>(y,x+xDlt+i) - fltt.at<uchar>(y,x-xDlt-i));
809       if( fltt.at<uchar>(y,x+xDlt+i) > fltt.at<uchar>(y,x-xDlt-i) )
810       { u += fltt.at<uchar>(y,x+xDlt+i) - fltt.at<uchar>(y,x-xDlt-i);
811       } else
812       { u -= fltt.at<uchar>(y,x+xDlt+i) - fltt.at<uchar>(y,x-xDlt-i);
813       }
814     }
815     if( u > v ) // robust++ ...
816     { x--;
817       break;
818     }
819   }
```

```
820   if( x == xc )
821   { for( x = xc-1; x > xc-xLong; x-- )
822     { int u = 0;
823       for( int y = yUp; y < yDown; y++ )
824       { for( int i = 1; i <= xLg; i++ )
825 //        { u += (fltt.at<uchar>(y,x+xDlt+i) - fltt.at<uchar>(y,x-xDlt-i)) *
826 //        (fltt.at<uchar>(y,x+xDlt+i) - fltt.at<uchar>(y,x-xDlt-i));
827         if( fltt.at<uchar>(y,x+xDlt+i) > fltt.at<uchar>(y,x-xDlt-i) )
828         { u += fltt.at<uchar>(y,x+xDlt+i) - fltt.at<uchar>(y,x-xDlt-i);
829         } else
830         { u -= fltt.at<uchar>(y,x+xDlt+i) - fltt.at<uchar>(y,x-xDlt-i);
831         }
832       }
833       if( u > v ) // robust++ ...
834       { x++;
835         break;
836       }
837     }
838   }
839
840 // exp (up&center+middle&border)
841 //     const int xk1 = 30;
842 //     const int xk2 = 2;
843 //
844 //     const int xk13 = 0;
845 //     const int yk11 = 3;
846 //     const int yk12 = 15;
847 //
848 //     const int xk23 = 20;
849 //     const int yk21 = 31;
850 //     const int yk22 = 40;
851 ////   for( int z = xc-xLong; z < xc+xLong; z++ )
852 //   for( int z = xc-xk1; z < xc+xk1; z++ )
853 //   { int u = 0;
854 //     for( int y = yk11; y < yk12; y++ )
855 //     { for( int i = 1; i <= xk2; i++ )
856 //         { u += sqrt((fltt.at<uchar>(y,z+xk13+i) - fltt.at<uchar>(y,z-xk13-
    i)) * (fltt.at<uchar>(y,z+xk13+i) - fltt.at<uchar>(y,z-xk13-i)));
857 ////      { if( fltt.at<uchar>(y,z+xk3+i) > fltt.at<uchar>(y,z-xk3-i) )
858 ////        { u += fltt.at<uchar>(y,z+xk3+i) - fltt.at<uchar>(y,z-xk3-i);
859 ////        } else
860 ////        { u -= fltt.at<uchar>(y,z+xk3+i) - fltt.at<uchar>(y,z-xk3-i);
861 ////        }
862 //       }
863 //     }
864 //     for( int y = yk21; y < yk22; y++ )
865 //     { for( int i = 1; i <= xk2; i++ )
866 //         { u += sqrt((fltt.at<uchar>(y,z+xk23+i) - fltt.at<uchar>(y,z-xk23-
    i)) *
867 //         (fltt.at<uchar>(y,z+xk23+i) - fltt.at<uchar>(y,z-xk23-i)));
868 ////      { if( fltt.at<uchar>(y,z+xk3+i) > fltt.at<uchar>(y,z-xk3-i) )
869 ////        { u += fltt.at<uchar>(y,z+xk3+i) - fltt.at<uchar>(y,z-xk3-i);
870 ////        } else
871 ////        { u -= fltt.at<uchar>(y,z+xk3+i) - fltt.at<uchar>(y,z-xk3-i);
```

```
872 ////         }
873 //       }
874 //     }
875 //     outfile << u << " ";
876 //   }
877 //   outfile << endl;
878 //
879 //   for( int y = yUp; y < yDown; y++ )
880 //   { fltt.at<uchar>(y,x) = 255;
881 //   }
882   fltt = fltt( Rect( cvRound(x-xWrF*wr), 0, cvRound(xWr2F*wr), fltt.rows ) )
      ;
883   resize( fltt, fltt, Size(32, 32), 0, 0, CV_INTER_LINEAR );
884
885 #endif // roi (TM/-)
886
887 #else  // not(_RI_BW) ---------------------------------- (tmp) BGR2FltBW-v.0
888
889   //-------------------------------------------------------------------| CHANGE
890
891   GaussianBlur( ft0, ft0, Size(7,7), 1.5, 1.5 );
892   flt = Mat::zeros( ft0.rows, ft0.cols, CV_8UC1 );
893   for( int y = 0; y < ft0.rows; y++ )
894   { for( int x = 0; x < ft0.cols; x++ )
895     { flt.at<uchar>(y,x) = saturate_cast<uchar>(
896         (uchar)(fltBp + fltPrm[0] * (float)ft0.at<Vec3b>(y,x)[0]
897                 + fltPrm[1] * (float)ft0.at<Vec3b>(y,x)[1]
898           + fltPrm[2] * (float)ft0.at<Vec3b>(y,x)[2]) );
899     }
900   }
901   //.............................
902 #endif // standard (_RI_BW)
903
904 #ifdef _SHOW_TIME
905   t1 = (double)cvGetTickCount() - t1;
906   printf( "%.2g\t", t1/((double)cvGetTickFrequency()*1000.) );
907 #endif
908 }
909
910
911
912
913
914
915
916 //-------------------------------------------------------------------------
917 // ************************************************************* change
918 int showImages( Mat& frame, Mat& fltt )
919 {
920
921 #ifdef _SAVE_ROI
922   static int nnn (1000);
923   char fname[50];
924   sprintf( fname, "roi/frame%.3d.jpg", nnn++);
```

```
925   imwrite( fname, fltt);
926 #endif
927
928 #ifdef _SHOW_IMAGE
929   Mat fltt_s;
930   const int xscale = 150,
931       yscale = 150;
932   resize( fltt, fltt_s, Size(150, 150), 0, 0, CV_INTER_LINEAR );
933   const int xs = 500,
934       ys = 200;
935   for( int y = 0; y < fltt_s.rows; y++ )
936   { for( int x = 0; x < fltt_s.cols; x++ )
937     { frame.at<Vec3b>(y+ys,x+xs)[0] = saturate_cast<uchar>(fltt_s.at<uchar>(
      y,x));
938       frame.at<Vec3b>(y+ys,x+xs)[1] = saturate_cast<uchar>(fltt_s.at<uchar>(
      y,x));
939       frame.at<Vec3b>(y+ys,x+xs)[2] = saturate_cast<uchar>(fltt_s.at<uchar>(
      y,x));
940     }
941   }
942   Point upLeft, downRight;
943   upLeft.x = xs;
944   upLeft.y = ys;
945   downRight.x = xs+xscale;
946   downRight.y = ys+yscale;
947 //  rectangle( frame, upLeft, downRight, CV_RGB(255,0,0), 3, 8, 0 );
948   rectangle( frame, upLeft, downRight, roiRectColor, 3, 8, 0 );
949
950   cv::imshow( "video", frame );
951   if(waitKey(10) >= 0) return -1;
952 #endif
953   return 0;
954 }
955
956
957 //---------------------------------------------------------------------
958 // ********************************************************* correct
959 void detect1( Mat& img, CascadeClassifier& cascade, double scale, int& xp,
      int& yp, int& wp, int& hp )
960 {
961   int i = 0;
962   Point pa, pb;
963   vector<Rect> faces;
964   Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale),
      CV_8UC1 );
965
966   cvtColor( img, gray, CV_BGR2GRAY );
967   resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
968   equalizeHist( smallImg, smallImg );
969 #ifdef _SHOW_TIME
970   double t = (double)cvGetTickCount();
971 #endif
972   cascade.detectMultiScale( smallImg, faces,
973     1.1, 10, 0
```

```
974     //|CV_HAAR_FIND_BIGGEST_OBJECT
975     |CV_HAAR_DO_ROUGH_SEARCH
976     //|CV_HAAR_SCALE_IMAGE
977     ,
978     Size(40, 40) );
979 #ifdef _SHOW_TIME
980   t = (double)cvGetTickCount() - t;
981   printf( "detection time = %.5g ms\n", t/((double)cvGetTickFrequency()
      *1000.) );
982 #endif
983   for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end(); r
      ++, i++ )
984   {
985     Mat smallImgROI;
986     vector<Rect> nestedObjects;
987     Point center;
988 //    int radius;
989 //    center.x = cvRound((r->x + r->width*0.5)*scale);
990 //    center.y = cvRound((r->y + r->height*0.5)*scale);
991 //    radius = cvRound((r->width + r->height)*0.25*scale);
992     xp = r->x*scale;
993     yp = r->y*scale;
994     wp = r->width*scale;
995     hp = r->height*scale;
996 //    circle( img, center, radius, CV_RGB(0,255,255), 3, 8, 0 );
997     pa.x = r->x*scale;
998     pa.y = r->y*scale;
999     pb.x = (r->x+r->width)*scale;
1000    pb.y = (r->y+r->height)*scale;
1001 //    rectangle( img, pa, pb, CV_RGB(255,255,255), 2, 8, 0 );
1002 //    rectangle( img, pa, pb, color, 2, 8, 0 );
1003  }
1004 }
```