ELSEVIER

Signal Processing: *Image Communication* 16 (2000) 333–352

SIGNAL PROCESSING: IMAGE COMMUNICATION

www.elsevier.nl/locate/image

Statistical model-based video segmentation and its application to very low bit-rate video coding

Huitao Luo[a,*], Alexandros Eleftheriadis[a], Jack Kouloheris[b]

[a]*Department of Electrical Engineering, Columbia University, MC4712, 1312 S.W. Mudd, 500 West 12th Str., 10027 NY, USA*
[b]*IBM T.J.Watson Research Center, NY, USA*

Received 12 February 1998

Abstract

This paper presents a statistical model-based video segmentation algorithm for typical videophone and videoconference applications. This algorithm makes use of online information to build and track statistical models for both the background and foreground on the fly. The segmentation algorithm is then rendered as a MAP problem. A hierarchical system structure is designed and spatial and temporal filters are used to improve the segmentation quality. The algorithm is implemented on a PC and runs in real time. In addition, two possible applications are discussed: generating video objects for the upcoming MPEG-4 standard and introducing subjective factors into the rate control of DCT-based coding algorithms. We focus on the second application by proposing a rate-distortion (R-D)-based optimal rate control algorithm for H.263. In this rate control algorithm, which we refer to as region-based rate control algorithm, previous segmentation results are used as subjective knowledge. A distortion model is created to integrate both subjective and objective factors and an R-D criterion is used to obtain the optimal bit allocation. With the proposed algorithm, an H.263 compatible encoder is implemented and it produces better perceptual quality in our experiments than standard H.263.

Keywords: Image coding; Image segmentation; Adaptive quantization; Real time processing

1. Introduction

In very low bit-rate video coding, many research efforts have recently been directed to utilizing the results from computer vision, especially segmentation. On one hand, computer vision techniques enable efficient use of the available bits in terms of subjective visual quality. Such techniques include topics like "region-based coding" [12], "model-assisted coding" [5] and "model-based coding" [2]. On the other hand, the development of multimedia brings about additional requirements on video coding, i.e., not only lower bit-rate, but also better functionality. The upcoming MPEG-4 standard also creates a major research problem: how to create video objects (VO)s efficiently?

General-purpose image segmentation has long been a difficult problem. The fundamental difficulty is that there is no necessary correspondence between low-level features such as color and edges and high-level *object* or *region* definition. Video segmentation is in some sense easier because it can make use of inter-frame motion information. Still

* Corresponding author.
E-mail addresses: luoht@ee.columbia.edu (H. Luo), eleft@ee.columbia.edu (A. Eleftheriadis), jacklk@watson.ibm.com (J. Kouloheris).

0923-5965/00/$ - see front matter
PII: S0923-5965(99)00062-4

there is no inherent association between *moving* regions and *video object* regions. When a moving object stops its motion, highly motion-dependent algorithms like [1,3,10] fail. Recently, there has been active research on morphological segmentation algorithms for region-based video coding [12,15]. However, their results produce only low-level regions without any semantic meaning and therefore, are still not satisfactory for MPEG-4 VO generation.

We propose a simple but interesting and effective online video segmentation algorithm, which uses an inexpensive videoconference quality camera and runs on Pentium PC platforms in real time. The key design issue in this work is to limit the application domain in the algorithm development and use some domain-specific knowledge to guide the segmentation. More specifically, in our case we assume that the algorithm only works on "head-and-shoulders" type videoconference images, in which the background is relatively stable and does not move rapidly (this is the most common case in videophone communication). Because of these limitations, the inherent difficulty of segmentation can be overcome in some degree.

This work is originally motivated by the human-body tracking work of "Pfinder" [17]. We use a similar statistical model to represent the background and foreground. However, we adapt their idea of "blob tracking" to video object segmentation. Unlike traditional computer vision approaches, the segmentation research in this work is carried out in the context of video coding. Accordingly, we emphasize real-time performance of the algorithm and try to evaluate the segmentation quality in the sense of video object coding. To meet these requirements, we introduce a hierarchical structure at the system level to integrate statistical models with spatial and temporal filters. With this design, video segmentation can be carried out in real time on average Pentium PCs with reasonable quality. This result makes it possible for us to combine the computer vision research with video coding research, and develop advanced video encoders that have the intelligence of high-level computer vision on low-end computers.

In the MPEG-4 framework, creating video objects is a straightforward application of our video segmentation algorithm. Even in traditional techniques like H.263, we also find application for our segmentation algorithm. From the segmentation result, we have a subjective idea of the input image before we actually encode it, i.e., we know which region is head and which are shoulders and background. Therefore, we can allocate more bits to the head region macroblocks (MBs) and less to those background MBs. This way we can use the bits more efficiently than a uniform MB encoding scheme that a standard H.263 encoder uses. There have been several similar papers [5,6,8] published under the name of *model-assisted coding*. Since our segmentation is improved, we can design better spatial and temporal adaptation in our bit-rate control algorithm. In addition, we design an integrated distortion model that includes both subjective and objective factors and try to optimize the rate control in the rate-distortion (R-D) sense.

This paper is organized as follows. In Section 2, we discuss blob-based statistical modeling of foreground and background, and the basic procedure of region classification and Kalman filtering for blob tracking. In Section 3, some system level implementation issues are discussed. A hierarchical system structure is developed based on the introduced statistical model to facilitate real-time performance. Related boundary relaxation and refinements are also discussed in this section. Section 4 describes the application of the segmentation results to an improved H.263 compatible encoder design. Region-based spatial and temporal quantization scalability is discussed, and experimental results are presented in comparison with Telenor's implementation of standard H.263. Finally, in Section 5 we conclude the paper.

2. Blob-based statistical region modeling and tracking

2.1. Motivation and context

This work is motivated by the widely used "chroma-key" technique and human body tracking work of "Pfinder" [17]. In addition, it is also related to the foreground/background segmentation work in [10]. The basic nature of the algorithm is

an online one. First, assume a background scene that contains no foreground, which enables the creation of a background model. Then, when the foreground enters, another model is created for the foreground. As discussed in the introduction, the purpose of modeling is to find proper representation of domain knowledge that helps in segmentation and tracking. Here we find it necessary to reiterate the context of our modeling algorithm. That is, the background of a videophone is not changing rapidly as compared with foreground motion and the foreground contains only one head-and-shoulders pattern.

2.2. Foreground model

In videophone cases, the foreground is "head-and-shoulders". We model it with two connected "blobs". Here the definition of "blob" is similar to that in [17], i.e., each blob has a spatial (x, y) and chromatic (Y, U, V) Gaussian distribution and a support map which indicates whether a pixel is a member of a blob. In this model, each pixel is represented by a feature vector (x, y, Y, U, V) and the feature vectors of the pixels belonging to the blob k have a Gaussian distribution with mean vector $\boldsymbol{m}_k$ and covariance matrix $\boldsymbol{C}_k$. Because of their different semantics, the spatial and chromatic distributions are assumed to be independent. That is, the matrix $\boldsymbol{C}_k$ is assumed to be block-diagonal.

In addition, for the clarity of succeeding discussion, we introduce some definitions related to the blob model as follows. First, the support map $s_k(x, y)$ for blob k is defined as

$$s_k(x,y) = \begin{cases} 1 & \text{if pixel } (x,y) \text{ belongs to blob } k, \\ & k = 1, 2, 3, \ldots, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Based on $s_k(x, y)$, blob k's *containing rectangle* rect_k is defined as the rectangle with the following coordinates:

$$\text{rect}_k = \text{rectangle}[(x_t, y_t), (x_t, y_b), (x_b, y_t), (x_b, y_b)], \quad (2)$$

where

$$\begin{cases} x_t = \sup_{s_k(x,y)=1}(x) \\ x_b = \inf_{s_k(x,y)=1}(x) \end{cases} \text{ and } \begin{cases} y_t = \sup_{s_k(x,y)=1}(y) \\ y_b = \inf_{s_k(x,y)=1}(y). \end{cases} \quad (3)$$

For segmentation purposes we also define a cumulative support map $s(x, y)$ for each image as

$$s(x,y) = \begin{cases} k & \text{if } s_k(x,y) = 1,\ k = 1, 2, 3, \ldots, \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

In addition, we define the foreground map $f(x, y)$ as the entire set of the support maps of foreground blobs

$$f(x,y) = \begin{cases} 1 & s(x,y) \neq 0, \\ 0 & s(x,y) = 0. \end{cases} \quad (5)$$

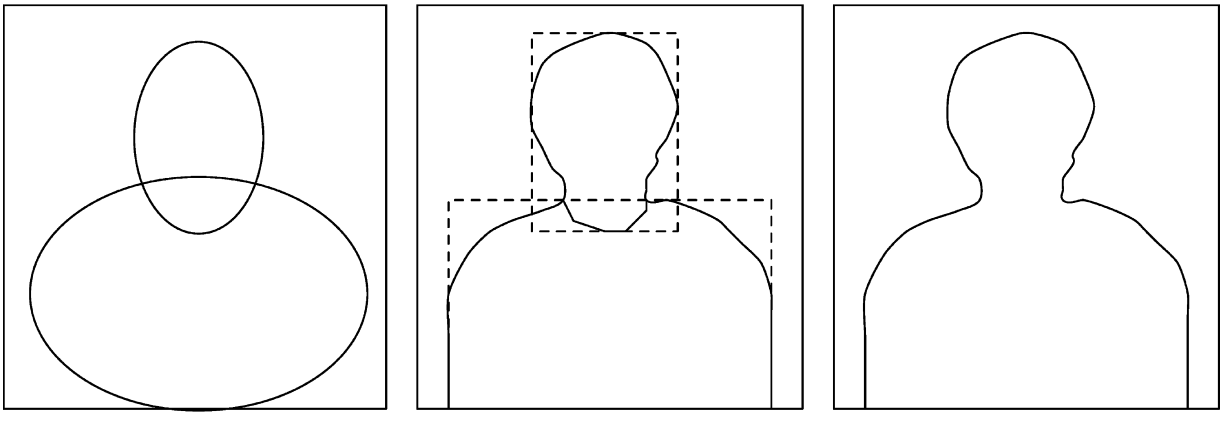

Fig. 1. Left: blob representation; middle: support map with containing rectangles; right: foreground map.

This relation can be demonstrated in Fig. 1. In Fig. 1, the left image illustrates two blobs, the middle image shows a support map containing rectangles and the right image is a foreground map.

The rationale behind blob modeling is that it represents an image region that has chromatic and spatial similarity. By the definition of a blob and its associated support map, low-level, pixel-oriented segmentation is associated with high-level, semantically meaningful blob tracking. In this way, high-level a priori knowledge can be used to guide low-level pixel segmentation.

2.3. Background model

The background is modeled as a texture map that varies over time. In common videophone applications, we assume the camera is static and there are no fast and major background changes. In this context, there are still several sources that may introduce temporal color variations in the background pixels and thus influence an accurate modeling. They are enumerated as follows.

1. The thermal noise of the camera sensor. This is mainly influenced by the quality of the camera. Two factors: noise stability and magnitude are related to our modeling work. In general, we tend to model it with a Gaussian distribution.
2. The gradual change of background for some reason. For example, a foreground human may move some items in the background or there may be changes in illumination, etc.
3. The automatic gain control (AGC) effect of the camera. This is quite obvious and disturbing when the foreground moves into or out of the scene or closer or farther away from the camera and changes the exposure of the camera. In these cases, the AGC mechanism introduces noticeable luminance variation in the background pixels even though the background itself does not change at all. Figs. 2 and 3 show our test results using Intel's Proshare videoconference camera, which does not have an AGC switcher. Fig. 2 is the histogram of the inter-frame variation of the luminance Y for background pixels when there is no foreground in the scene. Fig. 3 is the inter-frame Y variation of the same background pixels (not occluded by foreground) when a person enters the scene 3 meter away from the camera. We can see that the shift can be decomposed into two parts: a uniform shift and

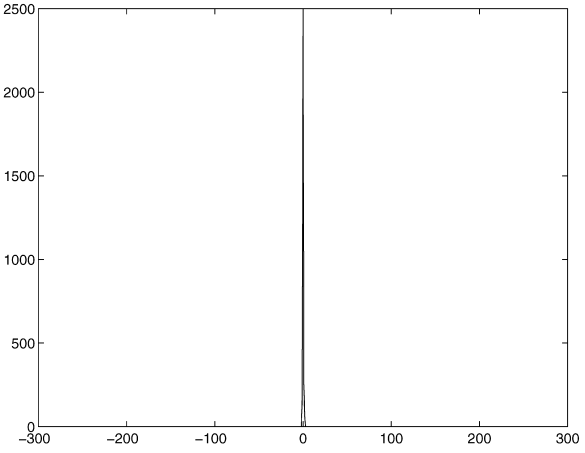

Fig. 2. Illustration of the AGC effects. When there is no foreground in the scene, the histogram of the visible background pixels' inter-frame luminance (Y) variance distribution is approximately a narrow Gaussian peak centered at zero. This is mainly caused by the thermal noise of the CCD camera.

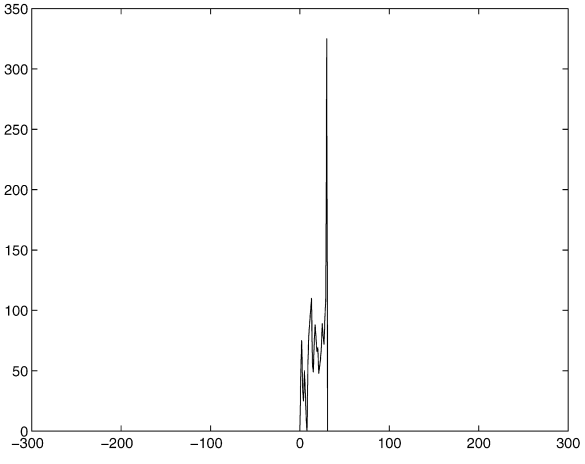

Fig. 3. Illustration of the AGC effects. When some foreground enters into the scene, the histogram of the visible background pixels' inter-frame luminance (Y) variance distribution can be interpreted as including two components: a uniform shifting part (corresponding to the highest peak in the right side) and a region adaptive part.

a pixel-dependent shift. According to our experiments, this effect is less important for those cameras with an AGC switch that can disable the AGC function. We have tested Sony's SSC S20 CCD camera with AGC function turned off and found that the foreground-induced background color shift can be ignored in most cases.

4. The shading effect brought about by the foreground motions. This factor can be compensated in part by introducing the normalized chromatic vector ($U^* = U/Y, V^* = V/Y$).

Given all these factors, we model each pixel in the background as a Gaussian distribution in the vector space (U^*, V^*) with mean vector $\boldsymbol{m}_0$ and covariance matrix $\boldsymbol{C}_0$. In the segmentation loop, the background model is created and updated as follows. First, the uniform shifting vector $\mathbf{diff}_t$ is estimated and compensated:

$$\mathbf{diff}_t = \frac{\sum_{f(x,y)=1}(\hat{\boldsymbol{y}}_t(x,y) - \hat{\boldsymbol{y}}_{(t-1)}(x,y))}{\sum_{(x,y)\in\mathscr{I}} f(x,y)}, \tag{6}$$

$$\boldsymbol{m}_{0,t}(x,y) = \mathbf{diff}_t + \boldsymbol{m}_{0,t-1}(x,y), \quad (x,y)\in\mathscr{I}, \tag{7}$$

where $\hat{\boldsymbol{y}}_t(x,y)$ and $\boldsymbol{m}_{0,t}(x,y)$ are the feature vector and model parameters for the current background pixel at the spatial position (x,y) and temporal position t (in succeeding references, $\hat{\boldsymbol{y}}_t, \boldsymbol{m}_{0,t}$ or $\hat{\boldsymbol{y}}$, $\boldsymbol{m}_0$ may be used when (x,y) and/or t information are not important), and vector $\mathbf{diff}_t$ is a frame level uniform shifting factor. As indicated in Eq. (6), $\mathbf{diff}_t$ is obtained by averaging the temporal feature vector difference over those background pixels that are not occluded by the foreground ($f(x,y) = 0$). However, the compensation in Eq. (7) updates every pixel within the background model, including those pixels occluded by the foreground as well (in Eqs. (6) and (7) $\mathscr{I}$ refers to the pixel set in one frame). This accounts for the uniform shifting effect of the above-mentioned third factor and is useful to model those background pixels that are uncovered by the foreground in motion.

In addition, each visible background pixel has its statistical parameters updated as

$$\boldsymbol{m}_{0,t} = \alpha * \hat{\boldsymbol{y}}_t + (1-\alpha) * \boldsymbol{m}_{0,t-1}, \quad 0 \leqslant \alpha \leqslant 1. \tag{8}$$

This compensates for the above-mentioned second, third and fourth factors.

Note that unlike the foreground model, each pixel of background is modeled individually. Or to be expressed in a uniform way, the feature vectors for the background model can also be put in the vector space (x, y, U^*, V^*) by implicitly including the spatial coordinate of each pixel (x,y). This approach can accommodate a variety of complex backgrounds without limiting them to fit to a structure like head-and-shoulders foreground.

2.4. Region classification

With the available statistical models for foreground and background, it is straightforward to classify pixels into different regions by their statistical likelihood. In our work, the *maximum a posteriori probability* (MAP) principle is used for the classification. We have two foreground classes (*shoulder and head*, $k = 1, 2$) and one background class ($k = 0$). To compensate for the shading effect, we choose to use feature vector $\hat{\boldsymbol{y}} = (x, y, U^*, V^*)$ instead of (x, y, Y, U, V) for foreground blobs as well (the major modeling principle remains the same). The logarithm likelihood that a feature vector $\hat{\boldsymbol{y}}$ belongs to blob k can be expressed as

$$\begin{aligned}\ln(p(\hat{\boldsymbol{y}}|\Omega_k)) = &-(\hat{\boldsymbol{y}} - \boldsymbol{m}_k)^{\mathrm{T}}\boldsymbol{C}_k^{-1}(\hat{\boldsymbol{y}} - \boldsymbol{m}_k) \\ &- \ln(\det(\boldsymbol{C}_k)), \quad k = 0, 1, 2,\end{aligned} \tag{9}$$

where Ω_k represents the event that the pixel belongs to class k. Based on MAP, each pixel is labeled in the support map as

$$\begin{aligned}s(x,y) &= \arg\max_k (\ln(p(\Omega_k|\hat{\boldsymbol{y}}))) \\ &= \arg\max_k [\ln(p(\hat{\boldsymbol{y}}|\Omega_k)) + \ln(p(\Omega_k))],\end{aligned} \tag{10}$$

where $\ln(p(\Omega_k)))$ is estimated based on typical videophone pictures.

We use two steps to convert the classified pixels into meaningful regions. First, the foreground pixels are processed with morphological filters to create a simple connected foreground map $f(x,y)$. Second, the support map $s(x,y)$ is obtained by blob growing, i.e., each blob is grown out within the support of the foreground map from their blob centers to create a simple connected support map. This is illustrated in Fig. 4.

2.5. Blob tracking

In previous sections we outlined the basic idea of the statistical model. To apply it to the video segmentation problem, blob models should be updated and tracked from frame to frame. Some initialization and controlling mechanisms are necessary. In this section, we discuss the basic tracking procedure in two loops: the model initialization loop and the model tracking loop.

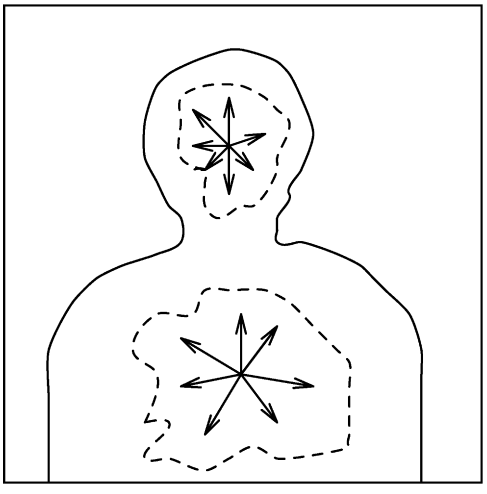

Fig. 4. Blob region growing illustration. Each blob's support map is grown out from their blob centers on top of the foreground map.

2.5.1. Initialization loop

The purpose of the initialization loop is to detect "head-and-shoulders" type foreground and create the foreground and background models. Its logical steps are illustrated in Fig. 5. At the beginning, only the background scene is captured and a background model is created. When a foreground enters, the system detects a model deviation and tries to analyze the size, speed and shape of the possible foreground and judge the likelihood of being a head-and-shoulders foreground. When a valid foreground is detected, a foreground model is created and the system enters the tracking loop.

2.5.2. Tracking loop

The flowchart of the tracking loop is shown in Fig. 6. As discussed in Section 2.4, the major steps of region segmentation are pixel classification, foreground morphological filtering and blob region growing. In addition to these steps, blobs are tracked with a Kalman filter. Unlike low-level pixel classification, blob tracking is carried out at the model-level with semantic meanings. Although in

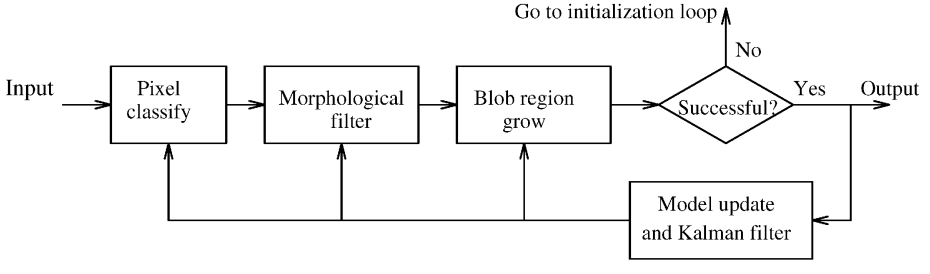

Fig. 5. Flowchart for initialization loop.

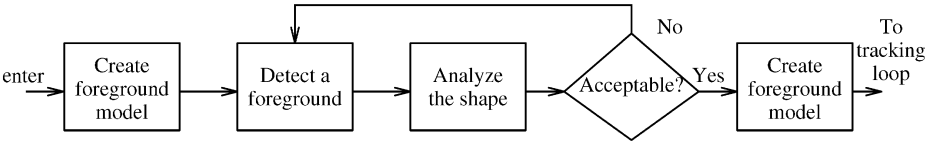

Fig. 6. Flowchart for basic tracking loop procedure.

this work the major purpose of segmentation is to get an accurate support map for each blob rather than to track the semantic information of blob motion, good modeling and tracking of blobs is still important because the tracked blobs carry the statistical model parameters, which are important for support map segmentation.

In this work, each blob is tracked independently with a Kalman filter. The observation vector for blob k includes the blob's center (x_k, y_k) and its statistical width and height (w_k, h_k):

$$\hat{\boldsymbol{Y}}_k = (x_k, y_k, w_k, h_k). \tag{11}$$

Here w_k and h_k are the variance of x_k and y_k. They can be obtained from the feature vector covariance matrix $\boldsymbol{C}_k$, i.e., $w_k = 2\sigma_{x_k}$ and $h_k = 2\sigma_{y_k}$. The dynamic model is a discrete Newtonian physical model of rigid body motion, which has the form

$$\hat{\boldsymbol{X}}(t + \Delta t) = \boldsymbol{\Phi}(\Delta t)\hat{\boldsymbol{X}}(t) + \xi(t), \tag{12}$$

where $\hat{\boldsymbol{X}}$ is the state vector, $\boldsymbol{\Phi}$ is the state transition matrix and ξ is the noise term. The $12D$ state vector $\hat{\boldsymbol{X}}$ and noise vector ξ contain four variables for the position of observation vector $\hat{\boldsymbol{Y}}$, four for the velocity and four for the acceleration, i.e.,

$$\hat{\boldsymbol{X}}(t) = \begin{pmatrix} \hat{Y} \\ V \\ A \end{pmatrix} \quad \text{and} \quad \xi(t) = \begin{pmatrix} \xi_{\hat{Y}} \\ \xi_V \\ \xi_A \end{pmatrix}. \tag{13}$$

From Newtonian physics, we have

$$\boldsymbol{\Phi}(\Delta t) = \begin{pmatrix} I & I\Delta t & 0 \\ 0 & I & I(\Delta t)^2 \\ 0 & 0 & I \end{pmatrix}. \tag{14}$$

In practice, the Kalman filter is used to predict the model parameters of each blob in the next frame, which is the starting point of region classification discussed in Section 2.4. In return, the result of region classification in the current frame is used to update the blob model parameters $\boldsymbol{m}_k, \boldsymbol{C}_k$, $k = 1, 2$, and the background model parameters $\boldsymbol{m}_0, \boldsymbol{C}_0$. In addition, it is also used as observation input to drive the Kalman filter for the next prediction.

Sometimes there are situations that the foreground moves too fast for the filter to follow or just moves out of the scene, and the system cannot find appropriate support maps at the predicted positions of the current frame. In these cases, as indicated in Fig. 6, the system automatically changes its status back to the initialization loop.

3. Hierarchical system design and implementation discussion

In the previous section, we discussed the blob-based statistical model and basic blob tracking procedures. In practice, we found out that updating the blob model parameters frame by frame involved expensive computation. In addition, because of the statistical nature of this segmentation algorithm, noise is inevitable and further filtering is always necessary to improve the segmentation quality. In order to solve these problems while limiting the overall computation complexity of the algorithm, we introduce a hierarchical structure at the system level.

3.1. Hierarchical architecture

With the new hierarchical design, the tracking loop of Fig. 6 is updated into Fig. 7. As indicated in Fig. 7, an input image is first subsampled by M in both horizontal and vertical directions. Model analysis and blob region tracking are carried out in the resultant lower resolution image. The processing result is then upsampled and further refined in the original resolution to produce the final output.

The benefits of this hierarchical structure come from two aspects. First, because the statistical model is tracked and updated in the lower resolution image, the computational complexity is reduced by M^2. Second, when the segmentation result in the lower resolution image is mapped back to the full resolution image, only the boundary blocks[1] are further processed by the following filters that are designed to suppress noise and improve the boundary quality. All the interior blocks may be skipped.

[1] One pixel in the lower resolution image is mapped into one M by M block in full resolution image.

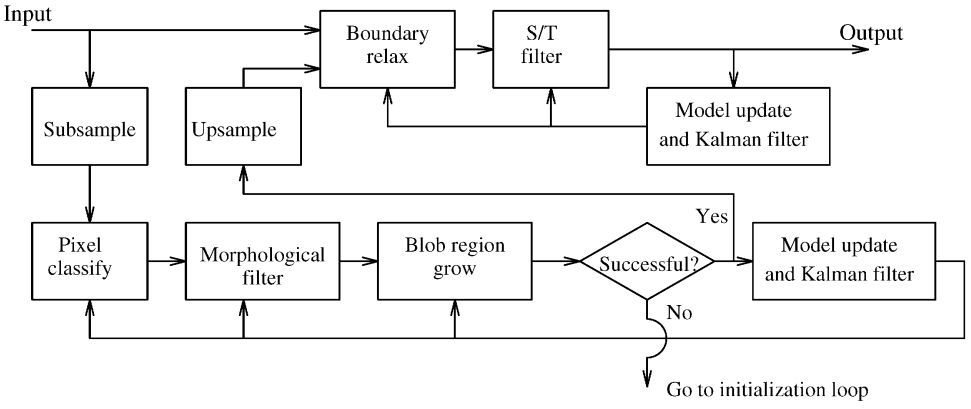

Fig. 7. Flowchart for hierarchical tracking system.

One drawback of the hierarchical structure is that we have to maintain two versions of the background model, one in the lower and one in the full resolution (for the foreground model we can maintain just one set of model parameters and convert them between different image resolutions). Compared with the benefits gained, this is not a big problem.

3.2. Boundary refinement

In an MPEG-4 context, we evaluate the segmentation boundary quality requirements in two aspects: spatial smoothness and temporal smoothness. They are addressed in the boundary relaxation module and the joint spatial and temporal median filter module, respectively. In general, we refer to these filters as S/T filters as shown in Fig. 7. Note that in this work, we only try to refine the foreground boundary. No efforts are made to refine the boundary between head-and-shoulder blob regions. For that part, we use their containing rectangle boundaries to get an approximation.

As the first step of boundary refinement, pixels in the boundary blocks are classified based on the foreground model and background model in the full resolution layer. After that, morphological filters are used to connect the segmentation results in each boundary block with interior block regions, so as to produce a simple connected foreground map.

After that, a relaxation procedure is carried out to improve the spatial smoothness of the foreground boundary. For this purpose, we use a boundary relaxation algorithm as introduced in [1]. The relaxation process examines each boundary pixel in foreground map $f(x, y)$, and sees if it should be flipped so as to improve the boundary smoothness. This can be converted to a statistical decision problem as follows.

First let Ω_k, $k = 0, 1$, represent the events $f(x, y) = k$, $k = 0, 1$. For each boundary pixel, its MAP classification function is

$$\ln(p(\Omega_k|\hat{\boldsymbol{y}})) = \ln(p(\hat{\boldsymbol{y}}|\Omega_k)) + \ln(p(\Omega_k)). \tag{15}$$

As discussed before, the first term on the right side can be obtained from Eq. (9). For the second term $\ln(p(\Omega_k))$, we define it to be a *smoothness measure*, which represents a priori knowledge, i.e., the more smooth a boundary it makes so that $f(x, y) = k$, the higher is $\ln(p(\Omega_k))$. Because smoothness is a spatial feature, we define the *smoothness measure* of a boundary locally for each of its boundary pixel in its 3×3 neighborhood as [1] does. The a priori density $p(\Omega_k)$ is modeled by a Markov random field considering a 3×3 neighborhood:

$$p(\Omega_k) = \frac{1}{Z} \exp\{-E(\Omega_k)\}, \quad k = 0, 1, \tag{16}$$

where Z is a normalizing factor and the energy term E is defined as

$$E(\Omega_k) = (n_B(k)B + n_C(k)C). \tag{17}$$

In Eq. (17), $n_B(k)$ and $n_C(k)$ are the homogeneity measure of the neighborhood if the current boundary pixel is labeled as k. They are obtained as follows. Each current boundary pixel constitutes eight *pixel-pair*s with its eight neighboring pixels. If both pixels in a pixel-pair have the same label, this pixel-pair is a *homogeneous pair*, otherwise it is a *heterogeneous pair*. n_B is the number of those inhomogeneous pairs that are in vertical or horizontal positions and n_C is the number of those in diagonal positions. B and C are two weighting factors that represent the distance factor of those pixel-pairs in different positions in relation to the boundary pixel under consideration. In general we should have $B = \sqrt{2}C$. The 3×3 neighborhood used for the boundary smoothness measure is illustrated in Fig. 8.

After the boundary relaxing, a seven-point 3D spatial-temporal median filter is used on the foreground map:

$$\mathrm{med}_t(x, y) = \mathrm{med}_7[I_{t-1}(x, y), I_{t+1}(x, y), I_t(x-1, y), I_t(x, y), I_t(x+1, y), I_t(x, y-1), I_t(x, y+1)]. \quad (18)$$

The purpose of this median filter is to suppress the temporal high-frequency noise on the boundary, which will be quite annoying when the segmented VOs are played back with an MPEG-4 player.

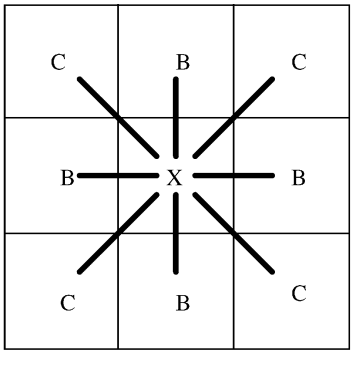

Fig. 8. Illustration of the 3×3 neighborhood used for boundary smoothness measurement in the boundary relaxation procedure. 'X' is the current boundary pixel. Eight thick bars represent eight pixel pairs. Inhomogeneous pixel pairs are numbered according to their positions: n_B is the number for those in the vertical/horizontal positions and n_C is the number for those in the diagonal positions.

Note that this filter introduces a delay of one frame. For real-time applications, a higher-order median filter is not desirable.

3.3. Segmentation experiments

We implemented the segmentation system on a 200 MHz Pentium PC with an Intel Proshare videoconference camera and capture card. The algorithm performance was 15 fps (frames per second) at QCIF size (176×144) and 30 frames per second at the sub-sampled size ($44 \times 36, M = 4$).

Because of the online feature of our segmentation algorithm, we could not use standard video sequences in our testing. Instead we captured a testing sequence with our videoconference system. The quality might not be as good as those standard sequences, but it is closer to the quality of real applications. The sequence is 800 frames in length, 10 fps, and consists of one person with considerable motion before a static background.[2]

Fig. 9 shows two segmentation results at the lower (a) and the full (b) resolution. Note that in image (b) the gray region labels the tracked head region while in image (a) the gray region is not used to enable better observation. In addition, the boundary between head-and-shoulders regions are approximated with their containing rectangle boundaries (for approximation purpose, the boundary for either rectangles may be used to separate head-and-shoulder regions). In Fig. 10, we show the effect of boundary relaxation and 3D spatial/temporal median filtering. In the first row from left to right are three consecutive original frames. The second row shows their segmented results without relaxation and filtering. The third row shows the final results with both relaxation and filtering. We can see that in the second row, some noise on the boundary is produced because part of the background's color is very similar to that of the foreground model. However, with boundary

[2] For public evaluation purposes, we put all the related experimental data of this paper on the web at http://www.ctr.columbia.edu/~luoht/research/region-based-h263.

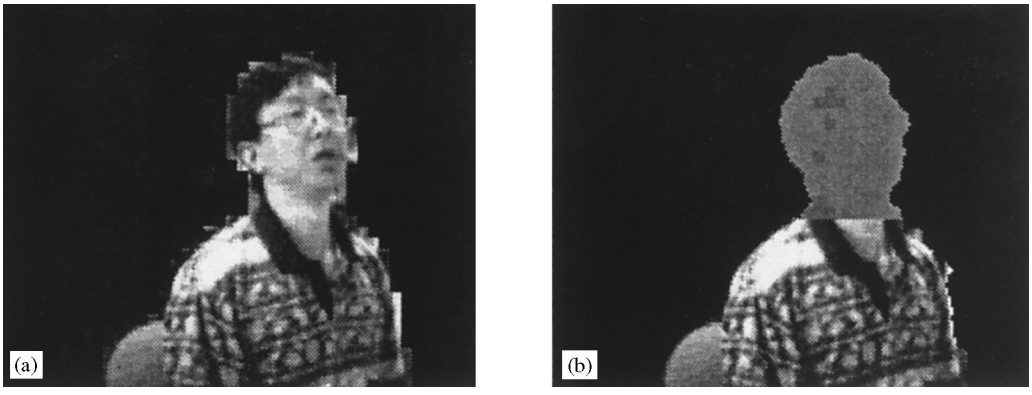

Fig. 9. Segmentation results at different resolutions: (a) the segmented result of the 260th frame of the testing sequence at the resolution $M = 4$; (b) the segmented result of the 260th frame at the full resolution. Note that in (b), the head region is labeled with gray color while in (a) the head region is not labeled in order to offer better observation. In addition, the boundary between head region and shoulder region is approximated with their containing rectangles.

Fig. 10. Comparison of filter effects. The first row from left to right is three consecutive original frames. The second row is their segmented result without spatial/temporal filtering and the third row is the final result with spatial/temporal filtering. We can see that the temporal high-frequency noise on the boundary is effectively removed.

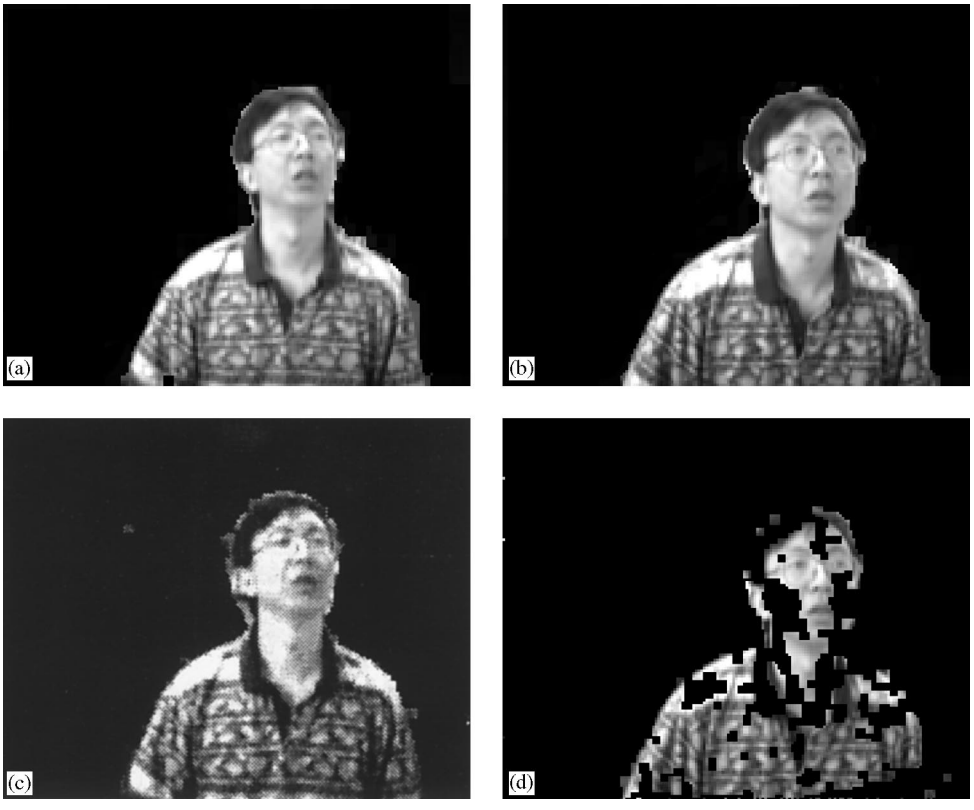

Fig. 11. Comparison of our model based segmentation algorithm with one of the proposed MPEG-4 VO segmentation algorithms (Hannover algorithm). (a), (b) are the segmentation results of the 233rd and 317th frame of the testing sequence by our model-based algorithm. (c), (d) are the segmentation results of the 233rd and 317th frame by the Hannover algorithm.

relaxation and 3D filtering, the boundaries in the third row are much smoother, both spatially and temporally. When played in real time, this high-frequency boundary noise will be visually quite annoying.

In order to evaluate our segmentation algorithm as a possible solution for MPEG-4 VO creation, we compared it with one of the proposed MPEG-4 segmentation algorithms [11]. [11] is a typical motion-based segmentation algorithm proposed by the University of Hannover, which we refer to as *Hannover algorithm* in this paper. Fig. 11 illustrates two typical frames of the segmentation results by our algorithm and the Hannover algorithm. (a), (b) are the segmentation results of the 213rd and 317th frame of the testing sequence by our model-based algorithm, and (c), (d) are the corresponding segmentation results by the Hannover algorithm.[3] Obviously, when the motion of the foreground is salient, Hannover algorithm produces as good a result as our model-based algorithm does, but when the foreground is relative static, the Hannover algorithm fails.

From the experimental results, we believe the discussed segmentation algorithm is useful for MPEG-4 video object creation for videoconference

[3] Because the Hannover algorithm has several a priori parameters, we tried to set them to the best of our knowledge. The parameters we used are: 5 for threshold for CDMi, 5 for constant L for CDMu and 5×5 neighborhood for morphological processing. For detailed meaning of these parameters, readers are referred to [11].

applications. The difference between this algorithm and algorithms like the Hannover algorithm is that our algorithm is specifically designed for videophone application while the Hannover algorithm is general purpose. But, as we have mentioned in the introduction, by limiting the application domain, we can get more domain knowledge and reduce the complexity of our algorithm. Only in this way we can apply the algorithm to real-time video services.

So far we have focused on MPEG-4 video object creation in our segmentation algorithm design. This segmentation algorithm is also useful for traditional DCT-based video coding techniques like H.263, because of its low complexity and real-time features. This is perhaps not so straightforward. The basic idea here is that from the segmentation result, we have a subjective idea of the input image before we actually encode it. We can then allocate more bits to the head region macroblocks (MBs) and less to the background MBs. This way, we can use the bits more efficiently than a uniform MB encoding that a standard H.263 encoder uses. In addition, for this application, only MB level segmentation is necessary and no pixel level accuracy is required. We can run only the subsampled ($M = 8$) part of the segmentation algorithm and no boundary refining in the full resolution is necessary, which makes real-time performance easier to achieve. In the following section of this paper, we describe in detail our design of a H.263 compatible encoder that makes use of segmentation results to improve its bit allocation and rate control.

4. Region-based bit allocation and rate control

4.1. The idea of encoder optimization

In the general framework of MPEG and H.261/H263 standards, there has been extensive research on optimal encoder design. This includes all the "non-standardized" steps such as coding model choice, bit allocation, adaptive quantization and rate control. The possible different choices of all these steps provide large room for optimization. In this work, we try to incorporate subjective factors, which are obtained with the previously discussed segmentation algorithm, into a traditional rate-distortion model and design an optimal H.263 compatible encoder in this sense.

Early relevant research can be found in [5,6,8]. In [6], an ellipse detection algorithm was used to detect the face region in typical head-and-shoulders videophone sequences and two techniques, i.e., buffer rate modulation and buffer size modulation were used in an H.261 compatible encoder to control the quantization that allocated more bits to the facial region. Lee and Eleftheriadis [8] extended this idea in two aspects: one was that it classified the image into four regions: eye, lip, face and background rather than two (face and background) as in [6] and assigned different subjective importance to each region. The other was that it introduced the idea of temporal scalability that used different temporal updating rates for different regions. For example, the lip region was updated at the highest rate in order to get lip synchronization. The ideas in these papers are interesting but we also find open problems. First, there is no uniform way to judge the rate-quality relationship and to choose proper trade-offs between rate and distortion in these rate control algorithms. Second, excessive segmentation of an image into multiple regions in these algorithms not only increases analysis complexity, but also brings visual artifacts. Applying different temporal rates to three different facial regions can, in some cases, be annoying when the decoded video is played back in real time.

Keeping all these in mind, we designed our region-based H.263 compatible encoder. We call it a *region-based encoder* in this paper. As a comparison reference, we first briefly introduce Telenor's H.263 Testing Model 5 (TM5) [7] in Section 4.2 and then describe our algorithm in Section 4.3.

4.2. Telenor's TM5

Telenor's implementation of TM5 supports most the coding models of H.263. Here we focus on its (online) rate control mechanism, which is implemented as follows.

1. The first intra picture is coded with $Q = 16$ (default value, can be set by user), and the buffer content is initialized as

$$\text{buffer} = R/F_{\text{target}} + 3R/\text{FR} \quad \text{and} \quad B_{t-1} = \bar{B}, \tag{19}$$

where R is the target bit-rate, FR is the frame rate of the source material (typically 25 or 30 Hz), F_{target} is target frame rate, B_{t-1} is the bits spent in the previous frame and $\bar{B}$ is the target number of bits for each frame (under uniform allocation).

2. For the following pictures the quantizer is updated at the beginning of each new macroblock line:

$$Q_{\text{new}} = \bar{Q}_{t-1}(1 + \Delta_G + \Delta_L),$$
$$\Delta_G = (B_{t-1} - \bar{B})/(2\bar{B}), \tag{20}$$
$$\Delta_L = 12\left(\sum_{k=1}^{i} B_{t,k} - (i/M)\bar{B}\right)\Big/ R,$$

where $\bar{Q}_{t-1}$ is the mean quantizer in the previous frame, M is the number of macroblocks in one frame, $B_{t,k}$ is the bits spent for the kth MB in the current frame (time t) and i is the index of current macroblock.

3. The buffer content is updated as the following pseudo-code:

```
buffer = buffer + B_t;
while( buffer > 3R/FR )
  { buffer - = R/FR; frame_incr + +; }.
```

Note that the variable `B_t` is just previous B_t and the variable `frame_incr` controls the number of frames skipped from the input video.

The simple designing consideration behind this scheme is that the available bits are allocated uniformly to every frame and every MB. A linear feed back is used to control the quantizer to make the actual bit consumption meet to the budget (in Eq. (20), Δ_G controls the bit allocation to frames and Δ_L controls the allocation to MBs). If a frame uses more bits than allocated, some succeeding frames are skipped in order to maintain the buffer content. Thus this scheme may not provide a fixed frame rate due to the frames skipped, though it accepts a target frame rate as an input.

4.3. Region adaptive bit allocation and rate control

In our work, we try to improve the rate control module of Telenor's implementation of H.263 while still maintaining the bit stream compatibility. More specifically, our contribution comes in three aspects: bit allocation, temporal adaptation and adaptive quantization.

4.3.1. Bit allocation

There are two steps in bit allocation: one is to allocate bits to frames and the other is to allocate bits to macroblocks. In H.263, there are only I and P frames but no B frames.[4] Because there is no random access requirement and H.263 is designed for real-time applications, P frames constitute the majority in an H.263 stream and I frames are only inserted to compensate for channel errors. It is not possible to pre-segment the input video into frame groups and assign bits to each I and P frame according to their relative complexity measures as in [14]. Therefore, in this work, we just allocate bits uniformly to each frame as TM5 does while concentrating on the controlling of bit allocation to different macroblocks.

Problem. The problem can be expressed as follows. Given the bit budget B, find the optimal bit allocation B_i to M macroblocks ($i = 1, 2, \ldots, M$) that minimizes the overall distortion in an R-D sense. As [9] pointed out, the bit budget B includes three portions: $B = B_s + B_m + B_c$, where B_s is the bits needed for header information, B_m is the bits needed to code the motion vectors, and B_c is the bits for DCT parameter coding. The portion of bits that we can control is only B_c. In [9], the bit allocation between B_c and B_m is also discussed (this means motion compensation should also be controlled). We do not consider this problem here. In the remaining part of this paper, B and B_i refer to DCT bits only.

Distortion model. In H.263, the DCT coefficients within a macroblock are quantized with one quantizer Q_i. Assuming that the DCT coefficients are distributed uniformly within one macroblock, we use the following expression to measure the coding

[4] Or more appropriately, *intra* and *inter* MBs, but in this work we only consider intra and inter transitions at the frame level. In addition, only baseline mode, no advanced modes like PB mode are considered in our work.

distortion for one frame:

$$D = \sum_{i=1}^{M} \beta_i (kQ_i^2), \tag{21}$$

where kQ_i^2 refers to the objective quantization errors with k being a constant and Q_i being the quantizer, β_i is a subjective importance factor for macroblock i. In this work, we decide β_i for each MB according to the support map $s(x, y)$ obtained through our segmentation algorithm, i.e., we assign higher β to those MBs labeled as head region and lower β to those labeled as shoulder and background regions.

Encoder model. In this work, we use the following equation to model the functional relationship between an H.263 encoder's bit consumption B_i and its quantizer Q_i:

$$B_i = a_i \sigma_i^2 Q_i^{b_i}, \tag{22}$$

where σ_i^2 is the (motion compensated) standard deviation of the luminance of the macroblock, a_i and b_i are two constants. According to our experiment as well as results reported in available papers [4,14,16], the empirical value of b_i is in the range of -1.5 to -2. In practice, we find $b_i = -2$ is a good compromise between computation complexity and modeling accuracy. In addition, because most motion estimotion modules produce standard absolute deviation (SAD) as a byproduct, we use mean absolute deviation (MAD) in place of standard deviation σ to save computation time. Here the definition of SAD and MAD are

$$\mathrm{SAD} = \min_{\mathrm{d}x,\mathrm{d}y} \left\{ \sum_{x=1}^{x=16} \sum_{y=1}^{y=16} |\mathrm{original}(x, y) - \mathrm{previous}(x + \mathrm{d}x, y + \mathrm{d}y)| \right\}, \tag{23}$$

$$\mathrm{MAD} = \frac{1}{256}\mathrm{SAD}, \tag{24}$$

where $(\mathrm{d}x, \mathrm{d}y)$ is the motion vector, and original(x, y) and previous(x, y) refer to pixel at the position (x, y) in the current and the previous frames, respectively. More specifically, we use

$$B_i = a_I \mathrm{MAD}_i^2 Q_i^{b_i} \tag{25}$$

to approximate previous Eq. (22) in our implementation. In our experiments, we find the behavior of Eq. (22) model and that of (25) are quite similar but the latter one is much easier to obtain. In addition, because both of them are empirical rather than theoretical, the accuracy of modeling also depends on proper choice and adaptation of model parameters like a_i, which we will discuss later.

R-D optimization. With distortion model Eq. (21) and encoder model Eq. (22), it is easy to solve the R-D optimization problem with Lagrange method:

$$S = D + \lambda B = \sum_{i=1}^{M} D_i + \lambda \sum_{i=1}^{M} B_i. \tag{26}$$

Setting $\partial S/\partial B_i = 0$, we have

$$\frac{\beta_i a_i \sigma_i^2}{B_i^2} = -\frac{\lambda}{k} = \text{constant}. \tag{27}$$

If we let

$$w_i = \sqrt{\beta_i a_i \sigma_i^2}, \tag{28}$$

then the optimal bits allocation to each MB can be obtained as

$$\hat{B}_i = \frac{w_i}{\sum_{k=1}^{M} w_k} \bar{B}, \tag{29}$$

where $\bar{B}$ is the average bits allocated to each frame. This equation shows that the optimal bit allocation $\hat{B}_i$ to MB i is proportional to its subjective important factor β_i and its objective complexity σ_i^2, which is in accordance with our intuition.

4.3.2. Temporal adaptation

In the bit allocation weight Eq. (28), the subjective factor β_i is used to control the bit allocation so as to introduce spatial adaptation. In most videoconference cases, the background is relative static and it is reasonable to reduce its temporal updating rate while still maintaining good visual quality. The benefit is that we can avoid spending bits on visually unimportant information. In some cases, for example, when the background itself is static, the bits we used for the background account for nothing but white noise. Even when there are changes in the background, it does not bring about much visual degradation to remove some temporal high frequencies in the background.

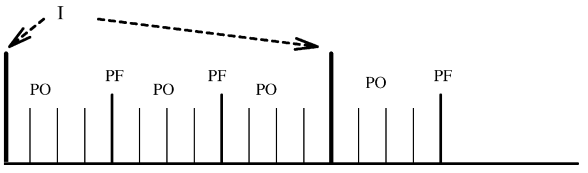

Fig. 12. Temporal scalability: PO and PF frames illustration. At the PO frames, only the foreground object's MBs are coded, while at the PF frames, all the MBs of the whole frame are coded. Thus the temporal updating frequency of the background is lower than that of the foreground.

Taking this into consideration, we classify the P frames in H.263 into PO and PF frames. In PO frames only the foreground object MBs are coded, while in PF frames all the MBs in the full frame are coded. PF frames work as anchors for PO frames and their temporal relation is illustrated in Fig. 12 where possible I frames are also included. The parameter T_{PF} (time between two PF frames) should be adjusted according to the estimation of background motion. In this work we measure T_{PF} as the number of PO frames between two adjacent PF frames. Also note that PO and PF control can be easily integrated into bit allocation Eq. (29) by assigning $\beta_{\mathrm{background}} = 0$ for PO MBs and $\beta_{\mathrm{background}} \neq 0$ for PF MBs.

4.3.3. Adaptive quantization

In the above discussion, we derived an optimal bit allocation to each MBs but how to choose proper quantizers to achieve the allocation remains a problem. Actually, since the encoder model is only an empirical one, there are no clear theoretical functions like $Q = f(B)$ or $B = f(Q)$. In addition, due to the real-time nature of H.263, it is not appropriate to introduce trial coding to estimate quantization model parameters as suggested in [4]. There should always be some online feedback mechanism in the rate control module in order to make sure that the actual bit-rate does not exceed the available bandwidth. In this work, we propose two adaptive quantization schemes to realize our bit allocation.

Scheme 1. Scheme 1 is developed based on Telenor's TMN5 model. We adapt Telenor's feedback rate control scheme as follows:

$$Q_{t,i} = \bar{Q}_{t-1,\,s(i)}(1 + \Delta_{\mathrm{G}} + \Delta_{\mathrm{L}}), \tag{30}$$

where

$$\Delta_{\mathrm{G}} = (B_{t-1} - \bar{B})/(2\bar{B}), \tag{31}$$

$$\Delta_{\mathrm{L}} = 4\left(\sum_{k=0}^{i} B_{t,k} - \sum_{k=0}^{i} \hat{B}_{t,k}\right)\Big/ R. \tag{32}$$

Compared with Telenor's scheme Eq. (20), we maintain different average quantizers $\bar{Q}_{t-1,\,s(i)}$ for different segmented regions $s(i)$. The idea is that the quantizers should be similar between neighboring frames for the same type of regions. Another change is that we use a different cumulative target rate $\sum_{k=0}^{i} \hat{B}_{t,k}$ to generate local adaptive factor, not a uniform allocation any more.

In the encoder model Eq. (22), parameter a_i is assumed to be MB-dependent. In the implementation of Scheme 1, we try to classify the MBs into different groups according to their complexity σ_i^2:

$$g(i) = \frac{\sigma_i^2}{G_{\mathrm{th}}}, \tag{33}$$

where G_{th} is a threshold to divide MBs into groups and $g(i)$ is the group index of macroblock i. The average a_i is tracked and updated for each group individually as $\bar{a}_{t,\,g(i)}$. When encoding MB i at the time t, we have

$$a_{t,\,g(i)} = \bar{a}_{t-1,\,g(i)}. \tag{34}$$

Note that in quantization model Eq. (22) we set b_i as -2 and only a_i is used to account for input adaptation. By grouping the MBs by their complexity and estimating model parameters individually for each MB group, we can better control the performance of this empirical model.

Scheme 2. Another way to control the quantizer Q_i is to make use of the empirical encoder model Eq. (22) directly. From Eq. (22), it is easy to derive the Q_i expression in relation to bit allocation B_i:

$$Q_i = \sqrt{\frac{a_i \sigma_i^2}{B_i}}. \tag{35}$$

So according to this empirical encoder model, we can allocate optimal B_i's as well as the Q_i's that are used to realize the optimal bit allocation. In practice, in order to fit the empirical model to actual input data distribution, we have to add a linear

feedback term and set the actual quantizer as

$$\tilde{Q}_i = \sqrt{\frac{a_i \sigma_i^2}{B_i}}(1 + \Delta_G + \Delta_L), \tag{36}$$

where the linear term $(1 + \Delta_G + \Delta_L)$ is as defined in Eq. (20). In addition, the encoding model parameter a_i is also adapted by groups as in Scheme 1.

Implementation issues. In H.263, the encoding quantizer can be adjusted at three layers: picture layer (QUANT), group of block (GOB) layer (GQUANT) and macroblock (MB) layer (DQUANT). Among them, both QUANT and GQUANT are 5-bit absolute values, while DQUANT is a 2-bit value that reflects the quantizer difference from the previous one. It is, therefore, not possible to set the quantizer of a macroblock arbitrarily as indicated in the previous two adaptation algorithms. The practical implementation is realized in a constrained domain if a bitstream compatibility is desired.

In our implementation, macroblocks are processed by GOBs. Each GOB contains one horizontal stripe of MBs. First for each MB i within the current GOB, its Q_i is calculated according to Eq. (30) or (35). An average quantizer is then obtained by

$$\bar{Q} = \sum_{i \in \mathrm{GOB}} Q_i \Big/ \sum_{i \in \mathrm{GOB}} 1. \tag{37}$$

The GOB quantizer is set to $\bar{Q}$ and DQUANT for each MB is then set in the best possible way.

In videoconference applications, the MBs in a GOB (a horizontal MB stripe) tend to belong to two classes: either the background and the head regions or the background and the shoulder regions. Because the background is assumed to be relative static, background MBs are skipped in most of the cases and they do not influence the GOB quantizer $\bar{Q}$ as defined in Eq. (37). Therefore, $\bar{Q}$ is obtained only for one class of MBs, either head region MBs or shoulder region MBs. These MBs tend to have similar quantizers and 2-bit DQUANT is sufficient to represent their difference.

4.4. Experimental results

We implemented a simulation version of our region based H.263 encoder on a PC with a hardware system as described in Section 3.3. As mentioned already, because of the online feature of our segmentation algorithm, we could not use standard video sequences in our testing. Instead we used the 800-frame testing sequence described in Section 3.3 and all the experiments in this paper were based on this testing sequence.

In the experiment, our region-based algorithm was compared with Telenor's TM5 [13]. In our algorithm, two different adaptive quantization schemes are used to realize the optimal bit allocation budget as discussed in Section 4.3.3. In the following discussion, we refer to these two algorithms as Scheme 1 and Scheme 2. We use *tmn-2.0* to represent Telenor's standard H.263 implementation because their latest software implementation is Version 2.0. In addition, all the three algorithms used in the experiments used the baseline mode, and no advanced modes are used. The target frame rate was 10 fps, and the bit-rate was set to 32 kbps (kilo bits per second). The parameters used in our algorithm were $G_{\mathrm{threshold}} = 200$, $T_{\mathrm{PF}} = 30$. The subjective factors used were $\beta_{\mathrm{head}} = 4$, $\beta_{\mathrm{shoulder}} = 1$, $\beta_{\mathrm{background}} = 0$ for PO frames and $\beta_{\mathrm{head}} = 1$, $\beta_{\mathrm{shoulder}} = 1$, $\beta_{\mathrm{background}} = 1$ for PF frames.

Fig. 13 shows the bit allocation of a typical PO frame by one of our region-based algorithms (Scheme 1). We number the images from left to right and top to bottom. Fig. 13(a) is an original frame (the 295th frame of the testing sequence), Fig. 13(b) is the gray scale display of the MAD (mean absolute deviation) of MBs (the brighter, the bigger), Fig. 13(c) is the bit allocation budget according to Eq. (29) and Fig. 13(d) is the actually achieved bit allocation by the adaptive quantization Scheme 1, which was discussed in Section 4.3.3. Note that (d) is not strictly equal to (c) due to the error in the empirical encoder model and the nature of the feedback control mechanism that is used to fit the model with the actual data.

To evaluate the encoder quality better, we introduce the concept of *region-based PSNR* and define it as

$$\mathrm{PSNR} = 10 \log_{10} \left\{ \frac{255^2 \mathscr{S}(s_k)}{\sum_{s_k(x,y)=1} [I(x,y) - \hat{I}(x,y)]^2} \right\},$$

where $I(x, y)$ and $\hat{I}(x, y)$ are the original and the decoded image pixel's scalar value (only luminance

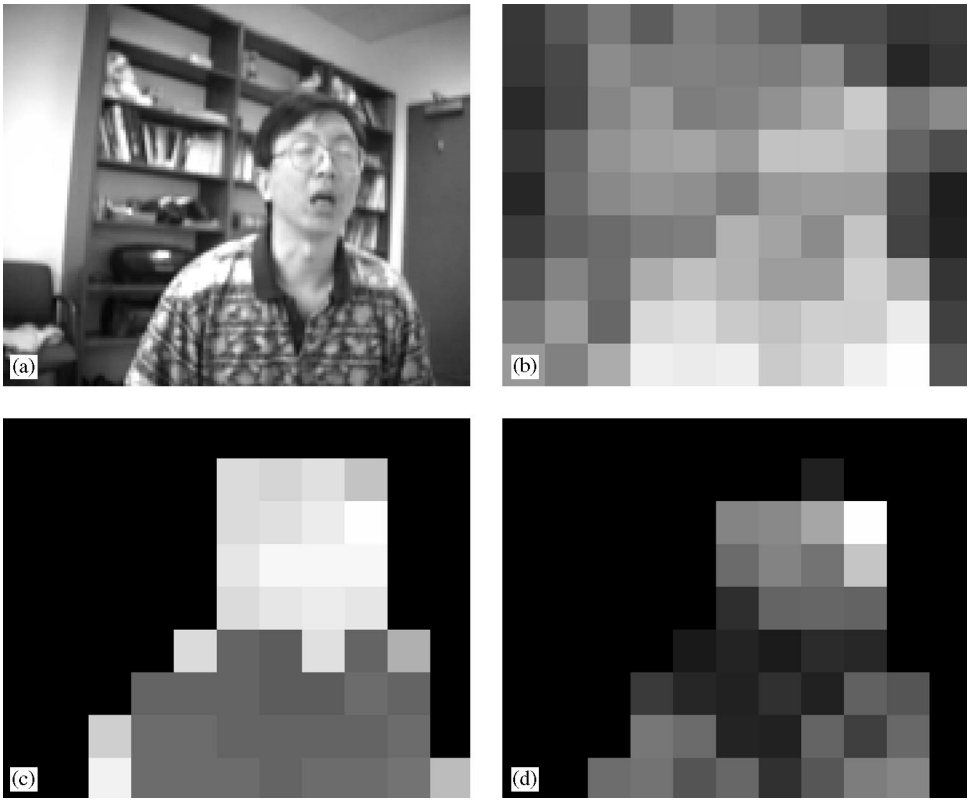

Fig. 13. Bit allocation of a typical PO frame by one of our region based algorithm (Scheme 1). (a) An original frame, (b) the grayscale display of the MAD (mean absolute deviation) of MBs (the brighter, the bigger), (c) the bit allocation budget according to Eq. (29) and (d) the actual achieved bit allocation by adaptive quantization Scheme 1.

factor Y is considered in this experiment), $\mathscr{S}(s_k)$ is the size of blob k's support map and can be defined as

$$\mathscr{S}(s_k) = \sum_{(x,y)\in\mathscr{I}} s_k(x,y).$$

With region-based PSNR, we try to compare the decoded image quality of our algorithm with that of the Telenor's standard H.263 algorithm. The testing sequence we used for explaining here is a 100-frame sequence (from the 220th to the 320th frame of the 800 frame testing sequence). To have a fair comparison of the rate control mechanism, we treat the decoded video frame by frame. That is, for those frames skipped by the encoder, their PSNR is calculated with the repeated previous frame at the decoder. Fig. 14 shows the frame-by-frame comparison of the head region PSNR of the three algorithms and Fig. 15 is the frame-by-frame comparison of the regular PSNR (calculated over the whole frame) of three algorithms. From these two figures, we see that our algorithms (both Scheme 1 and Scheme 2) improve the head region PSNR by about 1–1.5 dB in the whole testing frame range as compared with *tmn-2.0*. This gain comes at the expense of the loss of the regular PSNR of our algorithms by about 0.5–1 dB, as compared with that of *tmn-2.0*. The regular PSNR loss is due mainly to the PSNR loss in the background region, which is always the biggest region in the image and thus has the largest influence on the overall PSNR result. However, we also notice that within the comparing frame range, *tmn-2.0* skips more frames than our algorithms, which is reflected in the frequent negative peaks in its PSNR curve. In Fig. 16 the bit-rate produced by the three algorithms are

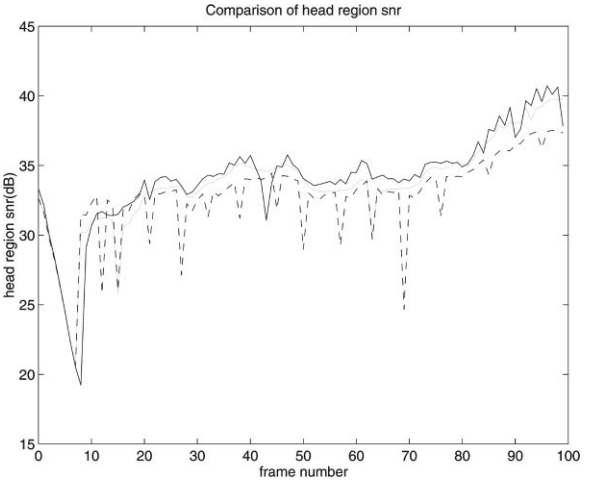

Fig. 14. Frame-by-frame comparison of the head region PSNR of the three algorithms. The light solid curve is for Scheme 1; the dark solid curve is for Scheme 2 and the dashed curve is for the *tmn-2.0* algorithm.

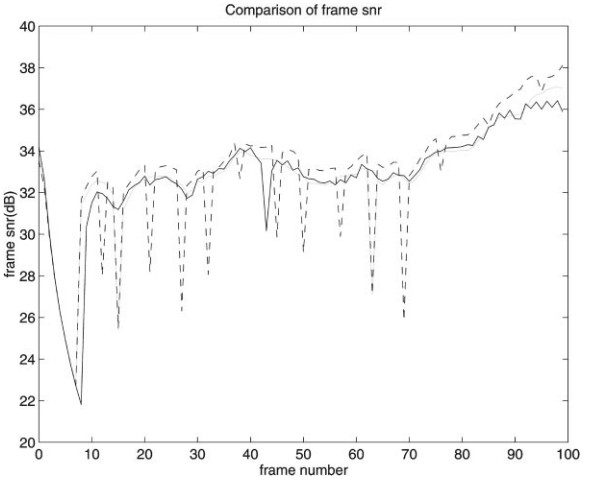

Fig. 15. Frame-by-frame comparison of the frame level PSNR of the three algorithms. The light solid curve is for Scheme 1; the dark solid curve is for Scheme 2 and the dashed curve is for the *tmn-2.0* algorithm.

compared. The curves indicate that both Scheme 1 and Scheme 2 produce smoother rate than *tmn-2.0*. This means the alternation of PO and PF frames scheme in our region-based algorithm does not bring about higher bit-rate fluctuation than *tmn-2.0*, and the feedback rate control scheme used

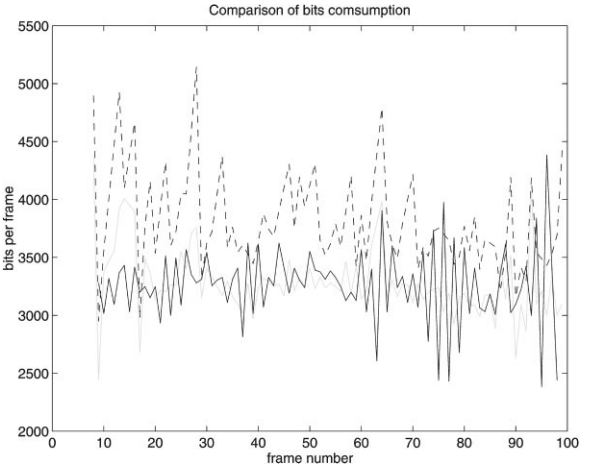

Fig. 16. Comparison of the bit consumption of the three algorithms. The light solid curve is for Scheme 1; the dark solid curve is for Scheme 2 and the dashed curve is for the *tmn-2.0* algorithm. The average bit-rate of *tmn-2.0* is higher than that of our two algorithms. That is because it skips more frames and thus has more bits to use for each coded frame.

works well. In Fig. 16, the average bit-rate of *tmn-2.0* is higher than that of our two algorithms. That is because it skips more frames and thus has more bits to use for each coded frame. In addition, when we compare our Scheme 1 and Scheme 2, we can see that Scheme 1 is a little bit better than Scheme 2 on the smoothness of the bit-rate produced, nevertheless their PSNR behaviors are quite similar.

To evaluate the PSNR performance more accurately, we list the frame-by-frame average of the PSNR within the 100-frame range for all the three algorithms in Table 1. In Table 1, we can see that both Scheme 1 and Scheme 2 encode about ten more frames than *tmn-2.0*. This means as a rate control algorithm, both Scheme 1 and Scheme 2 are better than *tmn-2.0*. In addition, due to their region adaptive bit allocation, both our algorithms produce higher head region PSNR than *tmn-2.0*, the difference is about 1 dB, averaged over the 100 frames. In the regular PSNR sense, our algorithms' performance is only about 0.2 dB below that of *tmn-2.0*. Unlike the regular PSNR curve in Fig. 15, the average regular PSNR drop of our algorithms is less because *tmn-2.0* skipped more frames, which penalized itself in the average performance

Table 1
Comparison of compression efficiency of our algorithm and that of the standard H.263. Rate is in kilo bit per second, PSNR is the peak SNR of the entire frame in dB, PSNR-b is PSNR of the background region, PSNR-h is PSNR of the head region and PSNR-s is the PSNR of the shoulder region

	Coded frames	Rate (kbits/s)	PSNR	PSNR-head	PSNR-s	PSNR-b
Scheme 1	91	31.84	32.86	33.66	29.72	35.75
Scheme 2	90	32.04	32.84	34.02	29.25	36.40
Tmn-2.0	79	32.26	32.98	32.75	29.94	36.51

Fig. 17. Comparison of a typical pair of reconstructed frames. (a) The reconstructed 260th frame (of the testing sequence) with Scheme 1; (b) the reconstructed 260th frame with *tmn-2.0*. The facial region of the left image (a) is clearer than that of the right image (b).

evaluation. Obviously, because of the relative amount of PSNR changes, we believe this result of our algorithm is surely interesting for most video-conference applications.

Fig. 17 compares two reconstructed frames by the region-based algorithm Scheme 1 (left) and *tmn-2.0* (right). The region-based algorithm exhibits better subjective quality in the head and facial region. In addition, the PO and PF alternation maintains decent background quality and the overall trade off is beneficial to the subjective quality evaluation.

Because we did not have source code for a real-time H.263 encoder, we did not implement a real-time system that combines the segmentation and H.263 encoding. In our experiment, we used Telenor's H.263 source code, which by itself is meant for simulation purpose and needs much optimization for a real-time performance. Thus an experimental result for the complexity of our region-based H.263 encoder is not yet available. However, as our segmentation algorithm runs at 30 fps for subsampled QCIF size video and most commercial H.263 software run at around 20 fps for QCIF video on average Pentium PCs, it is reasonable to believe that we can combine the segmentation and H.263 encoding into one system and still attain the frame rates around 8–10 fps for QCIF video. We expect to do this work in the near future.

5. Conclusion

In this paper, we proposed a simple online video segmentation algorithm for videophone applications. This algorithm uses feasible limitation on its application domain but gets good results with relatively low computational complexity. Because of its low complexity and real-time performance, this segmentation algorithm is especially useful for introducing some degree of intelligence into real-time video coding. One direct application of this

segmentation algorithm is MPEG-4 VO creation. Another application is to apply this segmentation algorithm to traditional DCT-based video coders. In our experiment, an H.263 compatible simulation encoder is implemented that makes use of the segmentation results in its rate control. Spatial and temporal adaptation are introduced and an improvement in subjective quality is observed. Our work shows that it is possible to combine the segmentation and traditional coding systems into an integrated intelligent coding system while still maintaining real-time performance on an average PC platform.

An interesting point worth noting is that we can use the discussed segmentation algorithm as *an optional module* for traditional videoconference systems. That is, when the segmentation module loses track or cannot fit the detected foreground to its internal blob model (for example, when there are multiple persons in the scene), it turns itself automatically back to the initialization loop and the H.263 encoder in the system works in its regular mode. However, when the segmentation module finds its expected foreground successfully and changes back to the tracking loop, the segmentation output is then available as an option to support region-based rate control for the H.263 encoder. This way, our segmentation algorithm can be incorporated into general-purpose videoconferencing systems without any concern that the tracking failure of the segmentation module will cause any negative problems to the whole system.

Acknowledgements

Part of this work was carried out during the first author's summer internship at IBM T.J. Watson Research Center. The authors would like to thank Dr. Cesar Conzales (IBM) and Prof. Dimitris Anastassiou (Columbia University) for their encouragement.

References

[1] T. Aach, A. Kaup, R. Mester, Statistical model-based change detection in moving video, Signal Processing 31 (1993) 165–180.

[2] K. Aizawa, Human facial motion analysis and synthesis with application to model-based coding, in: Motion Analysis and Image Sequence Processing, Kluwer Academic Publishers, MA, 1993.

[3] J.G. Choi, S.W. Lee, S.D. Kim, Spatio-temporal video segmentation using a joint similarity measure, IEEE Trans. Circuits Systems Video Technol. 7 (1997) 279–285.

[4] W. Ding, B. Liu, Rate control of MPEG video coding and recording by rate-quantization modeling, IEEE Trans. Circuits Systems Video Technol. 6 (1) (February 1996) 12–19.

[5] A. Eleftheriadis, A. Jacquin, Model-assisted coding of video teleconferencing sequences at low bit-rates, in: Proceedings of IEEE International Symposium on Circuits and Systems, London, England, June 1994, pp. 3.177–3.180.

[6] A. Eleftheriadis, A. Jacquin, Automatic face location detection for model-assisted rate control in H.261-compatible coding of video, Signal Processing: Image Communication 7 (1995) 435–455.

[7] ITU Telecommunication Standardization Sector LBC-95, Study Group 15, Working Party 15/1, Expert's Group on Very Low Bitrate Visual Telephony, Video Codec Test Model Tmn5, January 1995.

[8] J.B. Lee, A. Eleftheriadis, Motion adaptive model-assisted compatible coding with spatio-temporal scalability, in: SPIE Visual Communications and Image Processing Conference, February 1997, pp. 622–634.

[9] Y.-W. Lee, F. Kossentini, R. Ward, M. Smith, Towards MPEG4: An improved H.263-based video coder, Signal Processing: Image Communication 10 (1997) 143–158.

[10] C. Lettera, L. Masera, Foreground/background segmentation in videotelephony, Signal Processing: Image Communication 1 (1991) 181–189.

[11] R. Mech, P. Gerken, Automatic segmentation of moving objects, ISO/IEC JTC1/SC29/WG11, MPEG 96/1549, November 1996.

[12] P. Salembier, M. Pardas, Hierarchical morphological segmentation for image sequence coding, IEEE Trans. Image Process. 3 (September 1994) 639–651.

[13] Telenor, H.263 Software TMN2.0 (ftp://bonde.nta.no/pub/tmn/software/tmn-2.0.tar.gz.).

[14] E. Viscito, C. Gonzales, A video compression algorithm with adaptive bit-allocation and quantization, in: SPIE Visual Communications and Image Processing Conference, Boston, MA, November 1991, Vol. 1605, pp. 205–216.

[15] D. Wang, Unsupervised video segmentation based on watersheds and temporal tracking, IEEE Trans. Circuits Systems Video Technol. 8 (5) (September 1998) 539–547.

[16] J. Webb, K. Oehler, A simple rate-distortion model parameter estimation, and application to real-time rate control for DCT-based coders, in: Proceedings of IEEE International Conference on Image Processing, Santa Barbara, CA, October 1997.

[17] C. Wren, A. Azarbayejani, T. Darrell, A. Pentland, Pfinder, real-time tracking of the human body, Technical Report 353, MIT Media Lab, 1995. Also in IEEE Trans. Pattern Anal. Mach. Intell. 19 (7) (July 1997) 780–785.