Motion Compensated Two-link Chain Coding for Binary Shape Sequences

Zhitao Lu and William A. Pearlman

***Abstract*—In this paper, we present a motion compensated two-link chain coding technique to effectively encode 2-D binary shape sequences for object-based video coding. This technique consists of a contour motion estimation and compensation algorithm and a two-link chain coding algorithm. The object contour is defined on a 6-connected *contour lattice* for a smoother contour representation. The contour in the current frame is first predicted by global motion and local motion based on the decoded contour in the previous frame; then, it is segmented into motion success segments, which can be predicted by the global motion or the local motion, and motion failure segments, which can not be predicted by the global and local motion. For each motion failure segment, a two-link chain code, which uses one chain code to represent two consecutive contour links, followed by an arithmetic coder is proposed for efficient coding. The *motion success* segment is represented by the motion vector and its length. For contour motion estimation and compensation, besides the translational motion model, an affine global motion model is proposed and investigated for complex global motion. We test the performance of the proposed technique by several MPEG-4 shape test sequences. The experimental results show that our proposed scheme is better than the CAE technique which is applied in the MPEG-4 verification model [1].**

***Keywords*: Chain coding, Contour motion estimation, Object-based video coding, contour representation, contour matching, MPEG-4.**

I. Introduction

With the emergence of multimedia applications, functions such as access, searching, indexing and manipulation of visual information at the semantic object level, are becoming very important issues in research and some standardization efforts, such as MPEG-4. In MPEG-4, each object is represented by three sets of parameters, shape, texture, and motion so that the object can be encoded, accessed and manipulated in arbitrary shape. Among these three sets of parameters, shape information is crucial for object representation and object-based coding. In order to transmit the shape of an object efficiently, a large number of techniques have been proposed [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11].

According to the coding results, shape coding techniques can be classified into two categories: lossless coding and lossy coding. Lossless coding methods transmit exact shape information to the decoder, while lossy coding methods tolerate a certain degree of distortion in order to improve the coding efficiency. According to the shape representation during the coding process, the shape coding techniques are typically classified as block-based techniques and contour-based techniques. Block-based techniques use a binary image to represent the shape of the video object; this binary image is encoded block by block as in the conventional image coding technique. By using the same block size as in texture coding, the shape coding and texture coding can be combined to be processed in the block level which brings simple architecture for real implementation, such as hardware implementation. Contour-based techniques perform the compression along the boundary of the video object. A polygon or a contour is usually used to represent the shape of a video object. For these representations, distortion between the decoded and original shape information is easy and well defined. According to the specified distortion, a coding algorithm can achieve lossy and/or lossless coding.

Among the block-based techniques, the context-based arithmetic encoding (CAE) [1] is one of the most successful methods for binary image coding and is applied to the JBIG standard [12]. In the CAE method, pixels of an image are encoded in a predefined order, typically raster scan order. It is assumed that a high degree of local correlation exists in the shape image. Each pixel is encoded according to a conditional probability distribution that is conditioned upon its context – the value of pixels in a local neighborhood. This context is used to access a table containing probability distributions. The table is created by a training procedure prior to coding; it also can be adapted during the coding procedure in the case of the adaptive CAE. The shape and size of the neighborhood are represented by a template. The widely used templates for the *intra* and *inter* mode coding are shown in Fig. 1. The CAE has been adopted in the MPEG-4 verification model because it is well integrated into the current MPEG-4 texture coding scheme. It also has the benefit of a short processing delay because the shape image is processed macro-block by macro-block. However, the block size conversion in the MPEG-4 shape coding scheme, which applies the CAE technique, shows a visually annoying staircase effect [1].

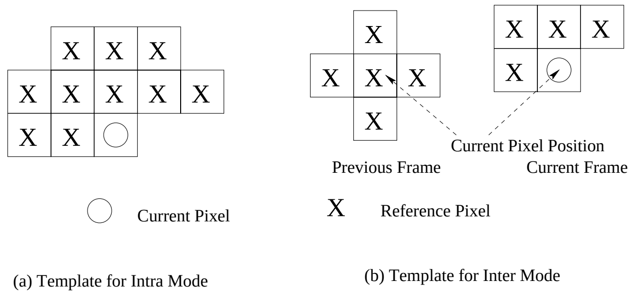

Fig. 1. Templates used in context-based arithmetic encoding

This work was performed at Rensselaer Polytechnic Institute and was supported in part by National Science Foundation Grant No. EEC-981276. The government has certain rights in this material.

Z. Lu is with Lucent Technologies, Whippany, NJ 07981, E-mail: zhitaolu@lucent.com

W. A. Pearlman is with the Electrical Computer and System Engineering Department, Rensselaer Polytechnic Institute, Troy NY 12180, E-mail: pearlw@rpi.edu

The chain coding method, first proposed by Freeman [2], is another widely used technique for shape coding which belongs to the contour-based technique. This method is based on the fact that successive points in a continuous contour are adjacent to each other. Instead of encoding the absolute position of each contour point, a *link* which represents the relative position between two consecutive contour points is encoded. In order to improve the coding efficiency of the chain code, a large number of schemes are proposed by imposing certain constraints on the contour to be encoded or by exploiting the spatial characteristics of contours [4], [6], [13], [14]. In [13], the contour link is defined on a 4-connectivity neighborhood structure. A constraint, that no contour points can be 4-connected to more than two others, is introduced to reduce the entropy of chain coding to 1.27 bits/link. Instead of imposing constraints on a contour, Kaneko et.al. proposes a segmented chain coding scheme to exploit the spatial redundancy within the object contour [4]. The whole contour is segmented into smooth segments in which links only have two neighboring directions. In each smooth segment, a 3-bit *Octant code* is used to represent the basic direction of this segment; the length of the segment is encoded by a variable length coder; and a series of 0 and 1 is used to represent the two neighboring directions. This scheme aims at the smooth contour, otherwise the overhead (length of the segment) is high.

Shapes in an image sequence generally change slowly or perhaps not at all from frame to frame, resulting in temporal redundancy that can be exploited to increase coding efficiency. Predictive contour coding schemes are proposed [3], [5]. A global motion and/or local motions are applied to predict the contour in the current frame based on the decoded contour in the previous frame. In these two schemes, the object contour is assumed to undergo rigid translational motion. Both global motion vector and local motion vectors are searched according to the number of matched contour points between two contours in the current frame and the previous frame. The whole contour is segmented into global motion success segments, local motion success segments and motion failure segments. Only those motion failure segments are encoded by the chain coding technique. This translational global motion model works well when the motion is translational motion, but it does not work well when the motion is more complex, such as zooming and/or rotation.

In this paper, we present a motion compensated two-link chain coding technique. The contour of the video object is defined on a 6-connected contour lattice for a smoother contour representation; a two-link chain coding technique is proposed to exploit the spatial redundancy within the object contour. In contour motion estimation and compensation, besides translational global motion model, we also investigate an affine global motion model for complex motion in the contour sequence. The paper is organized as follows. The contour lattice contour representation is introduced in section 2. In section 3, we present the two-link chain coding technique. In section 4, both translational and affine global contour motion estimation are presented. In section 5, we describe the whole shape coding system. The experimental results on MPEG-4 shape sequences are presented in section 6. Finally the paper is concluded in section 7.

II. Contour Representation

Our approach is a modified chain coding method which belongs to the contour-based technique. A contour-based coding method usually consists of two steps: extracting the contour from the binary shape image–label image, in which 1 is for object pixel and 0 otherwise; and encoding the contour. Before discussing the contour coding, we introduce the contour representation. Typically, a contour can be defined in two domains. One is in the original image pixel domain, where the shape image pixel is defined, and another is the half-pixel domain. In this paper, we use *image lattice* and *contour lattice* to represent these two domains.

In the image lattice, contour points are defined as the pixels with at least one differently labeled neighborhood. In the contour lattice, the contour points are defined in the half-pixel positions. As illustrated in Fig. 2, the $\circ$ represents the original image pixel; the $*$ and $+$ represent the half-pixel positions between two neighboring image pixels in the horizontal and vertical directions; and the box represents the remaining half-pixel positions. Throughout the whole paper, *top* position, *left* position and *vertex* position are used to refer to these three types of half-pixel positions.

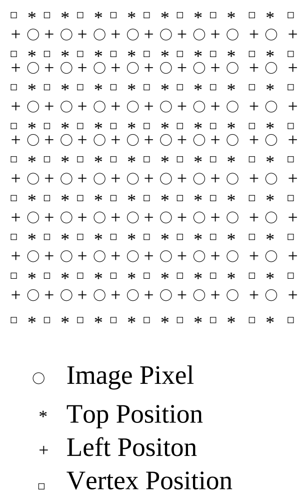

Fig. 2. Contour lattice

In our chain coding system, the contour is defined on the contour lattice and only on the *top* and *left* half-pixel positions. A contour point is on a *top* (or *left*) position when its two neighboring image pixels in the vertical (or horizontal) direction have different labels. By defining the contour in such a way, the number of links of a contour is identical in both image lattice and contour lattice representations. Our contour lattice is a 6-connected image. Two neighborhood structures centered at the *top* and *left* positions are shown in Fig. 3.

There are several advantages of this contour lattice representation over the image lattice representation. First, the contour lattice can handle more general contours. For example, if part of the object is one-pixel wide, there are difficulties in forming a closed contour on image lattice while it works well on the contour lattice at the same situation. Secondly, the contour lattice representation makes the contour smoother. As shown in Fig. 4, if the boundary of an object is in a diagonal direction, the contour in image lattice is *left, right, left, right, left, right, left, right*,

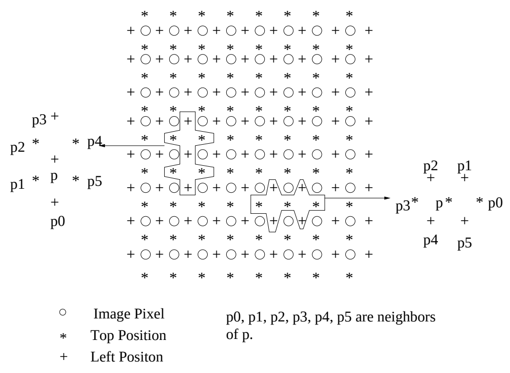

Fig. 3. Neighborhood structures of the contour lattice

while the contour in the contour lattice is *diag, diag, diag, diag, diag, diag, diag, diag*. The coding efficiency can be improved when an entropy coder is used.

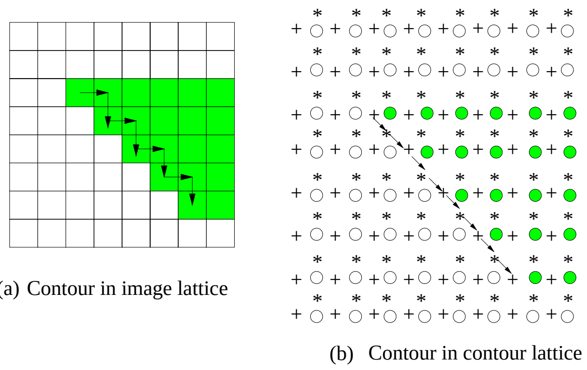

Fig. 4. Contour defined on the contour lattice is smoother than that on the image lattice

III. Two-Link Chain Code

In order to improve the contour coding efficiency, we impose constrains on the contour to be encoded by applying a majority filter [3]. A size one element structure of the majority filter we used is shown as in Fig. 5. We also used a FIFO queue to implement the filtering process which avoids the iteration in the orginal algorithm [15]. A majority filter simplifies the contour by changing the pixel under consideration to the same as the majority pixels within the element structure. The effect is similar to the perfect 8-connectivity constraint. We also encode two links per code to exploit the spatial redundancy of the contour.

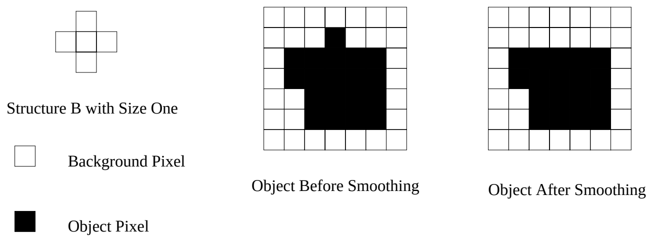

Fig. 5. Element structure and smoothing effect of the applied majority filter

As presented in Section II, contour points are defined on the *top* and *left* positions on a 6-connected contour lattice. From each position, there are 6 possible links to its six neighbors. After majority filtering, the number of the next possible links is more limited. As shown in Fig. 6, if the current *link* is in the horizontal or vertical direction, there are 3 possible directions for the next *link*. If the current *link* is in a diagonal direction, there are only 2 possible directions for the next *link*. All other directional links are not possible. As shown in Fig. 7, we analyze the impossible links when the last link is in a diagonal direction. If the next contour point is p_2, then image pixel i_1 belongs to an object while image pixels i_2, i_3, i_4, i_5, i_6 do not belong to that object. In this case, this object point will be eliminated during the majority filtering process, because more pixels (i_2, i_4 and i_6) within the structure element [3], [15] belong to the background. If the next contour point is p_3 or p_4, the last link will go to these points directly without passing p_c. The same analysis can be done when the last link is in the horizontal or vertical direction. The same result can be concluded if the last contour point is at a *left* position. As shown in Fig. 6, from each last contour point p_L, there are 6 possible directions to current contour point p_c. Two (2) out of 6 (1/3) of them are in horizontal direction; and 4 out of 6 (2/3) of them are in diagonal direction. If the direction is horizontal, there are 2 possible directions for next contour point; if diagonal, there are 3 possible directions for the next contour point. If we assume also that each type of *link* occurs with equal probability, entropy of the chain code under these constraints will be:

$$\frac{2}{3}\log_2 2 + \frac{1}{3}\log_2 3 = 1.19 \text{ bits/link}$$

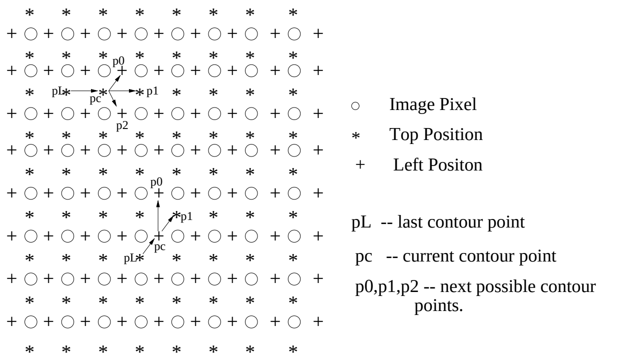

Fig. 6. Possible contour links in the proposed contour lattice

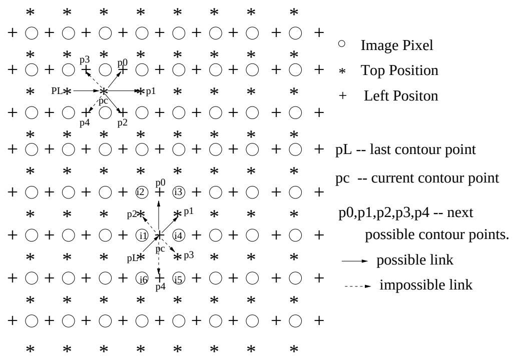

Fig. 7. Impossible contour links in the proposed contour lattice

Since there is spatial correlation among contour points, we use one chain code to represent two consecutive links. The final chain code is shown in Fig. 8. When the last link is in the horizontal or vertical direction, there are 7 possible combinations for the next two contour links. When the last link is in the diagonal direction, there are 5 possible combinations for the next two contour links. In order to reduce the bit-rate for the straight contour segment, we add two more dashed links. When the contour segment is a straight line in a diagonal direction, one code can represent 3 or 4 contour links. Without using an entropy encoder, the bit-rate of the proposed chain code is less than or equal to 1.5 bits/link.

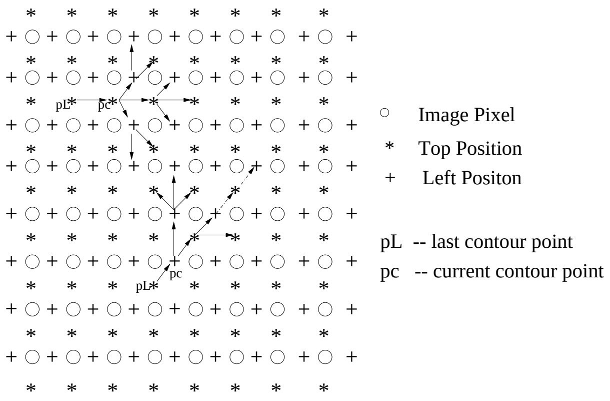

Fig. 8. Proposed two-pixel chain code

We use a context-based arithmetic encoder to encode the chain code sequence for a higher coding efficiency. The context is the direction of the last contour link. In total, there are 12 contexts shown in Fig. 9 and 10. Six of them begin from a *top* position, and the other six begin from a *left* position. Each number in the two figures represents a codeword for each contour link combination. The probability of each codeword under each context is adapted during the coding procedure.

IV. Contour Motion Estimation and Compensation

Since a contour sequence has very high correlation in temporal domain as the texture does, a straightforward method to exploit its temporal redundancy is using motion estimation and compensation. The contour in the current frame can be predicted from the contour obtained in the previous frame. The contour segments which can not be predicted are encoded by the shape coding technique. This can reduce the bit-rate of shape coding drastically. In the literature, the contour motion us usually assumed as translational motion. This approach works well for image sequences with low speed or simple motion. When there is zooming and/or rotation, this assumption does not work well. In this paper, we use two global motion models, a translational global motion model and an affine global motion model, to predict the global motion of the contour sequence to be encoded. In this contour based motion compensation, two kinds of contour segments, *motion success segment* and *motion failure segment*, are defined. After motion compensation, when all pixels within a contour segment in the current frame match all pixels within a motion compensated contour segment in the reference frame (previous frame), this segment is called *motion success* segment; otherwise, it is called *motion failure* segment. We define distance between two contour segments c_{n-1} and c_n as follows:

$$\text{dist}(c_{n-1}, c_n) = \max\{d(x_{i,n-1}, x_{j,n})\}, \quad (1)$$

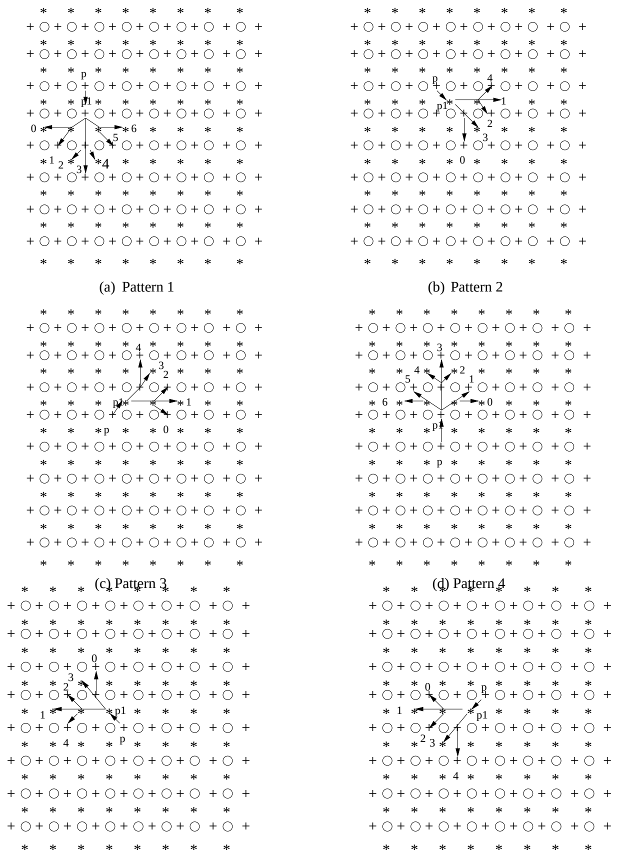

Fig. 9. 6 contour contexts and their chain codes beginning from the left position

$$(x_{i,n-1} \in c_{n-1}, x_{j,n} \in c_n)$$

where $d(x_{i,n-1}, x_{j,n})$ is the Euclidean distance. The *motion success segment* means the distance between motion compensated contour segment and the contour segment in current frame is zero; otherwise, it is a *motion failure segment*. This definition is applied on both the global motion compensation and local motion compensation. In the implementation, a threshold, which defines the minimum length of *motion success* segment, is applied in order to prevent segmenting object contour to very small segments. The overhead for a small *motion success* segment, its motion vector and length, makes it less efficient than the proposed chain coding method.

A. Translational Contour Motion Estimation

In this contour motion estimation/compensation scheme, the object contour is assumed to undergo a translational motion. A global motion vector for the whole object contour is searched according to the number of matched contour points between the object contour in current frame and the object contour in the previous frame under this global motion. The whole contour is then segmented into *global motion success* segments and *global motion failure* segments as shown in Fig. 11. For each *global motion failure* segment, *local motion vectors*, are searched. Each *global motion failure* segment is further split into *local motion success* segments and *local motion failure* segments. The *global motion success* segment can be represented by its length and the global motion vector. The *local motion success* segment can be

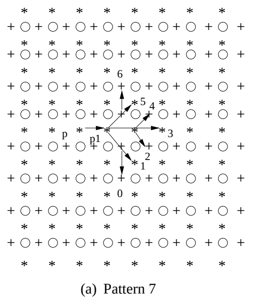
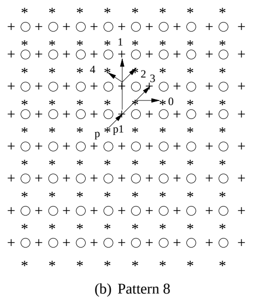
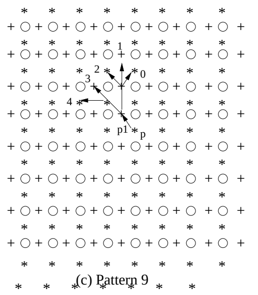
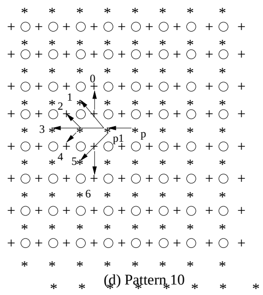
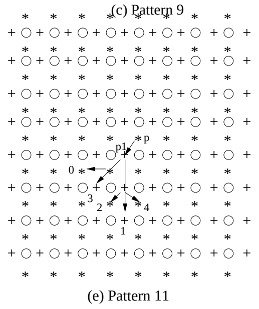
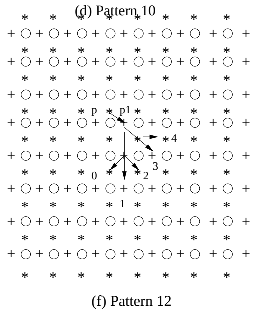

Fig. 10. 6 contour contexts and their chain codes beginning from the top position

represented by its length and the local motion vector. The *local motion failure* segment is encoded by the two-link chain code followed by an arithmetic coder.

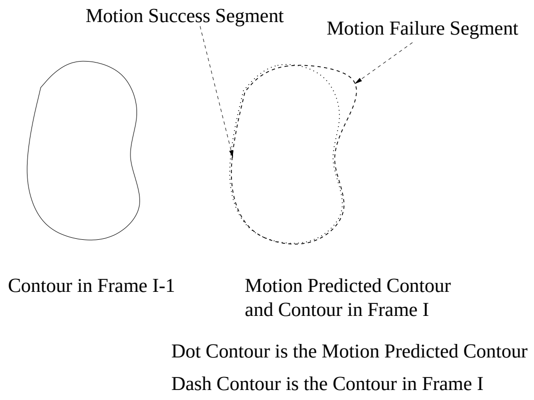

Fig. 11. Motion success contour segment and motion failure contour segment

B. *Affine Contour Motion Estimation*

When there is more complex motion such as *zoom* and/or *rotation*, the contour motion can not be well compensated by a translational motion model. Here we investigate an affine global motion model for these cases. We use the following six-parameter affine motion model as the global contour motion model.

$$\hat{x} = a_1 x + a_2 y + a_3 \tag{2}$$

$$\hat{y} = a_4 x + a_5 y + a_6 \tag{3}$$

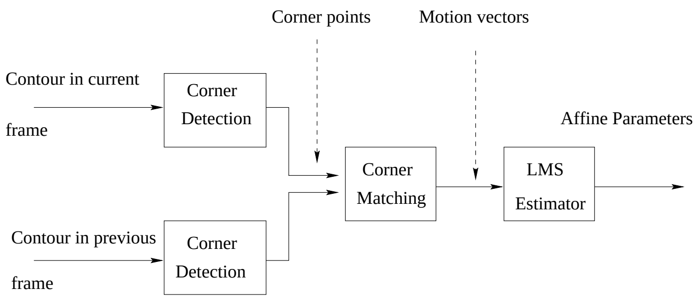

Fig. 12. Diagram of the affine motion estimation for a contour sequence

In the above equations, $\hat{x}$ and $\hat{y}$ are the coordinates of contour points in the current frame. x, y are the coordinates of contour points in the previous frame.

The problem is to estimate the vector $[a_1, a_2, a_3, a_4, a_5, a_6]$ according to available contours. Since the number of the contour points in the two contours is not necessarily equal, there is no unique solution to determine which contour point in the current frame corresponds to the contour point in the previous frame. Instead of using every contour point in these two contours, we only use some feature points on these two contours, corner points, to solve Eq. 3. The motion of these feature points can represent the motion of the whole contour, and the computational complexity is reduced significantly. The diagram of the motion estimation process is shown in Fig. 12. First the corner points of each contour are detected according to their curvature values. Then, the corner points are matched by a corner matching process [16]. The motion vectors are calculated from the matched corner pairs. The affine parameters are estimated by a least median square algorithm [17].

1) Corner Detection: The corner detection technique we used is from TargetJr package [18]. Corners on a contour are detected according to their curvature values. The algorithm is illustrated in Fig. 13. The curvature at a contour point i is defined as shown in Fig. 14:

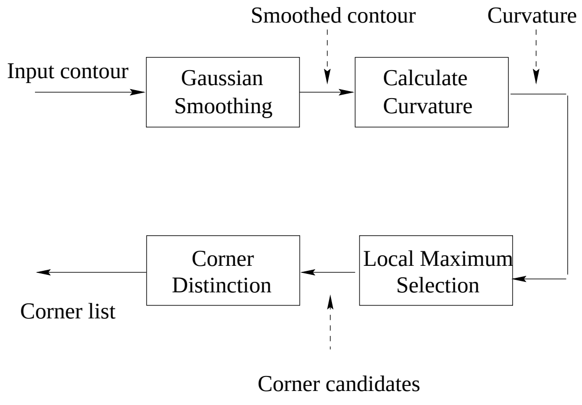

Fig. 13. Corner detection

$$\begin{aligned} &\text{curvature[i]} = 1 - \cos(\theta_i) \\ &= 1 - (\cos(\alpha_{i+1})\cos(\alpha_{i-1}) \\ &+ \sin(\alpha_{i+1})\sin(\alpha_{i-1})) \end{aligned} \tag{4}$$

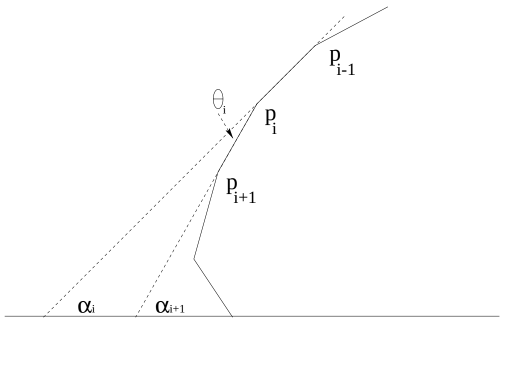

Fig. 14. Definition of the contour curvature

where $\cos(\alpha_{i+1})$, $\sin(\alpha_{i+1})$, $\cos(\alpha_{i-1})$, $\sin(\alpha_{i-1})$ can be calculated from the coordinates of contour points p_{i+1}, p_i and p_{i-1}. The larger the value of curvature in Eq. 4, the more likely the contour point p_i is a corner.

In order to overcome the digitization effect, the contour is first smoothed by a Gaussian filter. It is equivalent to convolve the contour with a Gaussian kernel. The curvature at each contour point is calculated according to Eq. 4. The contour point whose curvature value is a local maximum and larger than a threshold is classified as a corner candidate. A corner candidate is a corner when its direction distribution is significantly different from that of its neighboring candidates. As shown in Fig. 15, the curvature of contour points between two corner candidates, c_{i-1} and c_i, is treated as a random variable. Its mean value, β_i, is the curvature between two neighboring contour candidates c_{i-1} and c_i; the variance, var_i, is calculated from the curvature between contour points p_j, p_{j+1} and c_i. The curvatures of contour points between corner candidates c_i and c_{i+1} is treated as another random variable. Its mean value, β_{i+1}, is the curvature between contour candidates c_i and c_{i+1}; the variance, var_{i+1}, is calculated from the curvatures between contour points p_{j+2}, p_{j+3} and c_{i+1}. A scalar parameter, *peak separation* defined in Eq. 5 below, is used to represent the difference between the direction distribution at both sides of a corner candidate, c_i.

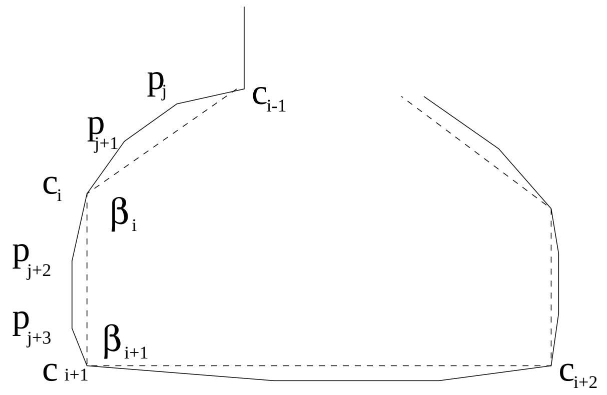

Fig. 15. Definition of the direction distribution of a contour segment

$$peak[i] = \frac{(\alpha_i - \alpha_{i+1})}{var_i + var_{i+1}} \tag{5}$$

The complete corner detection algorithm can be summarized as below:

1) Smooth the contour points with a Gaussian kernel.
2) Calculate the curvature value for each contour point.
3) Select the contour points whose curvature values are the local maximums and larger than the threshold as corner candidates.
4) Calculate the *peak separation* of each corner candidate.
5) Delete the corner candidate with the weakest *peak separation*, and merge the *direction distribution* of the contour segments connected to this corner candidate. Update the *peak separation* of the neighboring corner candidates.
6) Go back to the last step until the expected number of corners is reached.

2) Corner Matching: The corner matching algorithm we used is proposed in [16]. For each corner of the contour in the current frame, we choose several corners of the contour in the previous frame as its matching candidates. A matching probability is defined between each corner pair, the corner and one of its matching candidate. The pair with a matching probability higher than a threshold is identified as a matched pair. The initial matching probability between each pair of corners is determined by the pattern of branches connected to that pair of corners according to Eq. 10. At this point, the corner pair with an initial matching probability higher than the predefined threshold is classified as a matched pair; otherwise, an iterative relaxation procedure is applied. Additional matches based on supporting descriptors are gathered from a wider neighborhood of branches and corners. Specifically, the supporting descriptors for a given corner i in frame $I-1$ with a candidate corner j in frame I are:

1) The geometry of neighboring corners in frame $I-1$ appears to match that of the neighbors of corner j in frame I.
2) Motion vectors measured over the same neighborhood of corners appear to match.
3) The location of corner i relative to the previously matched corners in frame $I-1$ is like that of corner j relative to the matched corners in frame I.

The matching probability between corner i in frame $I-1$ and corner j in frame I is formulated as below. As illustrated in Fig. 16, there are $M = 4$ branches (l_0, l_1, l_2, l_3) connected to corner p_i in frame $I-1$ and $M = 4$ branches (l'_0, l'_1, l'_2, l'_3) to p'_j in frame I. They form 3 angles ($\theta_0, \theta_1, \theta_2$) and ($\theta'_0, \theta'_1, \theta'_2$) respectively. A cost function $C(i,j)$ expressing the match between corner i in frame $I-1$ and j in frame I is:

$$C(i,j) = w_L e_L(i,j) + w_\theta e_\theta(i,j) \tag{6}$$

where

$$e_L(i,j) = \sum_{i=1}^{M} |l_i - l'_i| \tag{7}$$

$$e_\theta(i,j) = \sum_{i=1}^{M-1} |\theta_i - \theta'_i| \tag{8}$$

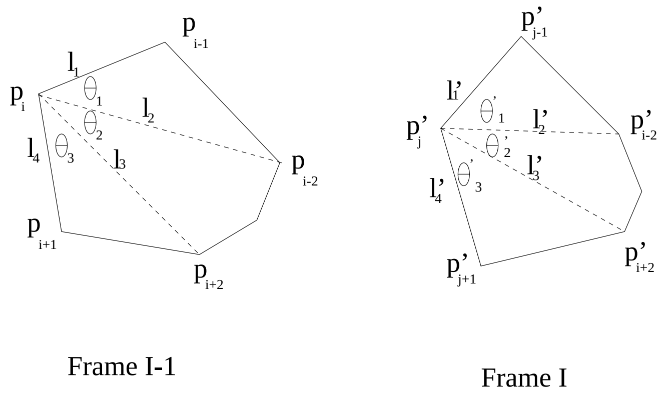

Fig. 16. The matching probability between a corner pair on two contours

Since $C(i,j)$ is small when the corner pair is close, we transform it to a matching score $S(i,j)$:

$$S(i,j) = (1 + C(i,j))^{-1} \tag{9}$$

Assume that there are M_1 corners in frame $I-1$. For each corner i, we choose M_2 corners in frame I which are the most nearest corners to i. The matching scores $S(i,j)$ are calculated for every pair of (i,j) where $i = 1,2,\ldots,M_1$ and $j = 1,2,\ldots,M_2$. The matching probability $P(i,j)$ is:

$$P(i,j) = \frac{S(i,j)}{\sum_{j=1}^{M_2} S(i,j)} \tag{10}$$

where $P(i,j)$ is a true probability obeying:

$$0 \le P(i,j) \le 1 \tag{11}$$

$$\sum_{j=1}^{M_2} P(i,j) = 1 \tag{12}$$

Three supporting descriptors can be formulated as below. The first one, $q_1(i,j)$, describes how well the corners connected to corner i (in frame $I-1$) match the corners connected to corner j (in frame I). Let corner i has N_1 connected corners, indexed $l_1 = 1,2,\ldots,N_1$. Similarly, corner j has N_2 connected corners, indexed $l_2 = 1,2,\ldots,N_2$. We try to match the groups of connected corners form frame $I-1$ with those in frame I by rotating one group relative to the other group until the best match is found, indicated by the maximum value of:

$$q_1(i,j) = \frac{1}{N'} \sum^{N'} P(l_1, l_2) \tag{13}$$

Where the combinations indicated by the right-hand argument (l_1, l_2) correspond to the possible rotations allowed in the matching process. $N' = \min(N_1, N_2)$.

The second supporting descriptor, $q_2(i,j)$, is the motion of the neighboring corners. Two neighboring corners of corner i in frame $I-1$ are chosen. Each of those two corners is compared with the M_2 closest corners in frame I. The most likely matched corner pair, indicated by the maximum matching probability in Eq. 10, are used to compute the corner motion vectors in x, y directions. These motion vectors are then averaged over the two test corners to give an average motion vector v_x, v_y. The motion discrepancy of the corner i in frame $I-1$ to the corner j in frame I is:

$$D(i,j) = \sqrt{(v_x(i,j) - v_x)^2 + (v_y(i,j) - v_y)^2} \tag{14}$$

The second supporting descriptor defined as:

$$q_2(i,j) = \frac{1}{1 + \mu D(i,j)} \tag{15}$$

The third supporting descriptor, $q_3(i,j)$, is defined as the matching probability by using the nearest matched corners to corner i. We form a reference corner network by connecting corner i to its three nearest matched corners. In frame I, the three counterparts of the matched corners form the network of corner j. $q_3(i,j)$ is defined exactly analogous to $S(i,j)$ in Eq. 9.

The three supporting descriptors are combined to give a matching update factor $Q(i,j)$:

$$Q(i,j) = \frac{\sum_{m=1}^{3} q_m(i,j)}{1 + \sum_{m=1}^{3} \sum_{j=1}^{M_2} q_m(i,j)} \tag{16}$$

This matching update factor is then used to update the matching probability of each pair of corners. The matching probability in iteration $k+1$ time, $P^{k+1}(i,j)$, is:

$$P^{k+1}(i,j) = \frac{P^k(i,j)(1 + Q(i,j))}{\sum_{j=1}^{M_2} P^k(i,j)(1 + Q^k(i,j))} \tag{17}$$

The matching probability of a real matched corner pair will be increased by this update factor and that of a non-matched corner pair will be decreased. When a corner pair has a matching probability higher than the threshold, it is classified as a matched pair.

The corner matching algorithm can be summarized as below:

1) For each corner point i in frame $I-1$, choose the M_2 nearest corners in frame I as its matching candidates.
2) Calculate the initial matching probabilities $P^0(i,j)$ for each corner pair (i,j). $k = 0$.
3) If $P^k(i,j)$ is higher than the predefined threshold, denote (i,j) as a matched pair.
4) For each corner which is not matched, calculate its supporting descriptors, update its matching probability. $k = k + 1$.
5) go back to step 3 until all corners are matched.

3) Affine Parameters Estimation: After we get all matched corner pairs, we can calculate the motion vectors between these matched corner pairs. The motion vectors are then used to estimate the affine parameters by the least median square (LMS) algorithm [17]. The LMS algorithm is robust to the data with outliers. Since not every corner's motion is consistent with the affine model, the LMS algorithm can avoid the effect of these corners by treating them as outliers.

In order to test the corner matching and the affine motion estimation algorithms, we create shape images undergoing translational and affine motion from an available shape image. As an example, we create a new shape image by moving the shape image in frame 0 of the Akiyo sequence with motion vector (5,-5).

Then we create another shape image by an affine motion with affine parameters $[0, -0.055, 0, 0.055, 0, 0]$. This corresponds to turning the shape image centered at (0,0) with an angle of 3.2 degree. The actual motion parameters and the estimated motion parameters are listed in Table I. The number of correctly predicted contour points by the decompressed motion parameters are listed in Table II. We can see that the estimated affine parameters are fairly accurate. When the object contour is under an affine motion, the affine model can compensate the motion better than the translational model.

TABLE I
TRUE AND ESTIMATED AFFINE PARAMETERS

	a_1	a_2	a_3
True	0.	-.055	-5.
Estimated	0.0017	-0.0535	-5.07
True	0.	0.	-5.
Estimated	0.	0.	-5.
	a_4	a_5	a_6
True	.055	0.	5.
Estimated	0.0592	-0.0007	3.90
True	0.	0.	5.
Estimated	0.	0.	5.

TABLE II
NUMBER OF PIXELS WHICH ARE CORRECTLY PREDICTED

	Corrected Pixels	Total Pixels
Translational	129	514
Affine	408	514

V. PROPOSED MOTION COMPENSATED CHAIN CODING SYSTEM

There are two coding modes: the *intra* mode and the *inter* mode in our proposed shape coding scheme. The contour in the first frame within a GOP (group of pictures) is encoded by the *intra* mode. The contours in other frames are encoded by the *inter* mode. The diagrams of the *intra* mode and the *inter* mode contour coding are shown in Fig. 17 and Fig. 18 respectively. In the *intra* mode coding, the contour is encoded directly by the chain coding method proposed in III followed by an arithmetic coder. In the *inter* mode coding, global motion parameters and local motion vectors are first searched. For global motion success segments, only the length of the segments are transmitted. For local motion success segments, the length and local motion vectors are transmitted. For motion failure segments, chain coding is applied.

Since the required resolution of affine parameters to be transmitted is high, it consumes more bits to transmit affine parameters than translational motion vectors. In real implementation, each affine parameter required 10 bits. The total number of bits required for affine parameters are 60 bits compared with the 10 bits required for translational motion vectors. Therefore we have two modes for global motion estimation: the *affine* mode

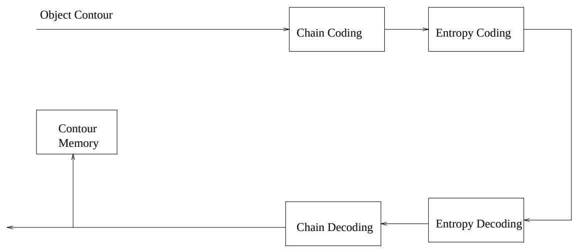

Fig. 17. Diagram for the intra mode contour coding

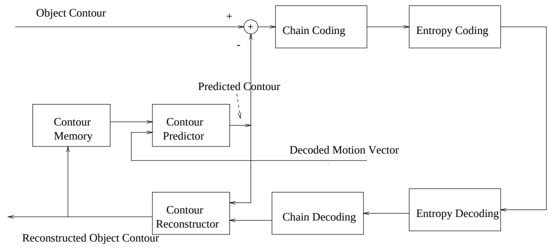

Fig. 18. Diagram for the inter mode contour coding

and the *translational* mode. When non-translational motion is significant, the number of correctly predicted contour pixels is lower. Then we turn on the *affine* mode. The syntax of the bitstream is shown in Fig. 19. In the bitstream, the first bit is the global motion flag, 1 for affine motion, 0 for translational motion. Following is the affine parameters or the translational global motion vector depending on the global motion flag. Then the coordinate of the start point (10 bits for x and 10 bits for y) is followed by the bitstream for each contour segment. In each contour segment, the first two bits represent the type of this segment. For a global motion success segment, only the length of the segment is transmitted. For a local motion success segment, the length of the segment and the local motion vector are transmitted. For a motion failure segment, a series of chain codes follows. An *END* is used to tell the decoder that the end of this segment is reached.

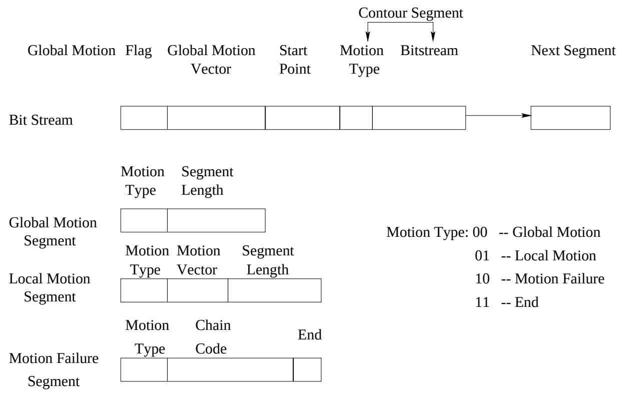

Fig. 19. Bitstream syntax for contour coding

VI. EXPERIMENTAL RESULTS

We test the performance of the proposed algorithm by coding several widely used MPEG-4 test shape sequences: the Akiyo and Weather sequences in QCIF and CIF format with frame rate of 30 fps. The total length of the sequences is 300 frames. We code the sequences in both the *intra* mode and the *inter* mode. First the shape image is transformed into an object contour defined on the *contour lattice*. Then the contour links are encoded by the proposed chain coding technique.

For the *intra* mode coding, the average bits used for coding each frame of these sequences are listed in Table III. From the data in Table III, the coder reached the bit-rate at 0.8-1.0 bits/link for lossless contour coding.

TABLE III
AVERAGE BIT USAGE FOR INTRA MODE CONTOUR CODING

Sequence	Bit Used	Total Links	bits/link	bits/pixel
Akiyo(QCIF)	454	508	0.89	0.0484
Akiyo(CIF)	857	1022	0.84	0.0228
Weather(QCIF)	425	447	0.95	0.0749
Weather(CIF)	795	918	0.863	0.0351

We compare the performance of our proposed shape coding technique with the CAE and baseline-based coder [7]. The results are listed in Table IV. From Table IV, the performance of our algorithm is better than that of the CAE and the baseline-based technique.

TABLE IV
COMPARISON OF SHAPE CODING TECHNIQUES IN INTRA MODE

Sequence	Frame	Format	CAE †	Baseline †	Proposed algorithm
Akiyo	0	QCIF	0.06059	N/A	0.0576
Akiyo	0	CIF	0.03657	N/A	0.0260
Weather	0	QCIF	0.05581	N/A	0.0574
Weather	30	QCIF	0.0801	0.0745	0.0579
Weather	0	CIF	0.03167	N/A	0.0252

†Algorithm does not require a contour smoothing before encoding.

In the *inter* mode coding, we apply the proposed technique on the same sequences. The average bits used for each frame of these sequences are listed in Table V.

TABLE V
AVERAGE BIT USED FOR INTER MODE CONTOUR CODING

Sequence	Bit Used	Total Links	bits/link	bits/pixel
Akiyo(QCIF)	179	508	0.35	0.0191
Akiyo(CIF)	469	1022	0.46	0.0125
Weather(QCIF)	288	447	0.63	0.052
Weather(CIF)	620	918	0.67	0.0276

We also compare the performance of our proposed shape coding technique with the CAE and another motion compensated contour-based technique, GPSC [5]. The results are listed in Table VI. From Table VI, the performance of our algorithm is better than that of the CAE and GPSC technique.

TABLE VI
COMPARISON OF SHAPE CODING TECHNIQUES IN INTER MODE
(BITS/FRAME)

Sequence	Format	frame rate	CAE †	GPSC	Proposed algorithm
Weather	QCIF	30 fps	303	N/A	288
Weather	QCIF	10 fps	382	394	356

†Algorithm does not require a contour smoothing before encoding.

VII. CONCLUSION

In this paper, we present our new two-link chain coding method for 2-D shape coding. The contour points are defined on the *contour lattice*, a 6-connected image. We impose a smooth constraint on the object contour by applying a majority filter to improve the coding efficiency. We test the performance of the proposed technique in several MPEG-4 shape sequences. In the intra mode, the bit-rate of 0.8-1.0 bits/link can be reached for lossless coding. In the inter mode, we investigate the translational and affine-model based motion compensated chain coding scheme for shape sequence coding. The corners on a contour are detected as the feature points. A corner matching algorithm is used to match corresponding feature points between contours. The affine global motion parameters are estimated from the motion vectors at the feature points. The experimental results show that affine motion model works well for more complicated global motion than the traditional translational motion model. The experimental results show that our proposed scheme uses fewer bits than that of the CAE technique which is applied in MPEG-4 VM7.0.

REFERENCES

[1] N. Brady, F. Bossen, and N. Murphy, “Context-based arithmetic encoding of 2d shape sequences,” in *Special session on shape coding, ICIP97*, 1997.

[2] H. Freeman, “On the encoding of arbitrary geometric configurations,” *IRE Trans. Electron. Comput.*, vol. 10, pp. 260–268, June 1961.

[3] C. Gu and M. Kunt, “Contour simplification and motion compensated coding,” *Signal Processing: Image Communications (Special Issue on Coding Techniques for Very Low Bit-Rate Video*, vol. 7, pp. 279–296, Nov. 1995.

[4] T. Kaneko and M. Okudaira, “Encoding of arbitrary curves based on the chain code representation,” *IEEE Trans. on Communications*, vol. COM-33, pp. 697–706, July 1985.

[5] J. I. Kim, A. C. Bovik, and B. L. Evans, “Generalized predictive binary shape coding using polygon approximation,” *Signal Processing: Image Communication*, pp. 643–663, July 2000.

[6] L. Labelle, D. Lauzon, J. Konrad, and E. Dubois, “Arithmetic coding of a lossless contour-based representation of label images,” in *Proc. of the International Conf. on Image Processing*, vol. 1, 1998.

[7] S. H. Lee, D.-S. Cho, Y.-S. Cho, S. H. Son, E. S. Jang, J.-S. Shin, and Y. S. Seo, “Binary shape coding using baseline-based method,” *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 9, pp. 44–58, Feb. 1999.

[8] F. Marques, J. Saueda, and T. Gasull, “Shape and location coding for contour images,” in *Proc. Picture Coding Symposium*, 1993.

[9] H. Musmann, M. Hotter, and J. Ostermann, “Object-oriented analysis-synthesis coding of moving images,” *Signal Processing: Image Communications*, vol. 1, pp. 117–138, Oct. 1989.

[10] P. Salembier, F. Marques, and A. Gasull, “Coding of partition sequences,” *Video Coding: The Second Generation Approach (L. Torres and M. Kunt, eds.)*, 1996.

[11] N. Yamaguchi, T. Ida, and T. Watanabe, “A binary shape coding method using modified mmr,” in *Special session on shape coding, ICIP97*, 1997.

[12] I.-T. R. T. 82, *Information Technology – Coded Representation of Picture and Audio Information - Progressive Bi-Level Image Compression*. ITU.

[13] M. Eden and M. Kocher, “On the performance of a contour coding algorithm in the context of image coding. part i:contour segment coding,” *Signal Processing*, vol. 8, pp. 381–386, July 1985.

[14] T. Kimoto and Y. Yasuda, “Highly efficient coding scheme for digital drawings based on their perfect 8-connectivity (in japanese),” *Trans. IECE Japan*, vol. J66-D, pp. 872–879, July 1983.

[15] Z. Lu, *New Coding Systems for Wavelet Transforms of Signals*. PhD thesis, Rensselaer Polytechnic Institute, Dec. 2000.

[16] R. N. Strickland and Z. Mao, “Computing correspondences in a sequence of non-rigid shapes,” *Pattern Recognition*, vol. 25, pp. 901–912, Sept. 1992.

[17] P. Rousseeuw and A. Leroy, *Robust Regression and Outlier Detection*. New York, NY: John Wiley Sons, 1987.

[18] “http://foto.huf.fi/research/targetjr/manpages,” in *TargetJr Package Webpage*.