A NOTE ON MERGING LINE SEGMENTS WITH THE SEARCH SPACE REDUCED BY A CONDITION BASED ON AN ORDERING

Leszek Chmielewski
EPSILON *Applied Research Group Co. Ltd.*
Daniłowiczowska 11/66, PL 00-084 Warsaw

Abstract. The problem of merging line segments into more meaningful, longer segments involves deciding which segments to merge. The full solution space would be searched if each segment were checked with each other one for possible match. The entire process should be repeated if segments merged in a previous step were many. For some types of segments, such an ordering relation can be found that only for the segments which are close to each other in this order the merging is probable. The condition of closeness in the order can be used to significantly reduce the search space. This simple observation is independent of the segment type and of the form of a sufficient condition for merging. Examples of merging linear segments are presented.

Key words: merging, search space reduction, line segments.

1. Introduction

Detection of lines in digital images can be considered on various levels:

1. **Pixel level.** Detection of pixels in which an edge occurs.
2. **Edgel level.** Grouping edge pixels into edge elements (*edgels*).
3. **Segment level.** Forming longer segments of edgels.
4. **Line level.** Merging segments into longer, more meaningful lines.

In the following paper, an observation which can help to simplify the operations on the line level will be considered.

Grouping line elements to form meaningful descriptions is a vital problem in numerous applications. The most frequently used technique seem to be the Hough transform, in its numerous versions ([4, 5, 11, 15, 16]; see [8] for a review). Grouping with the use of various perceptual properties of lines is also studied (see e.g. [3, 9, 10, 14]). This problem can be treated together with the problem of segmenting curves into meaningful parts [1, 6]. A good example is [12], where the splitting and merging operations are performed in sequence.

When merging of any kind of objects is considered, the question arises, *which objects should be merged* at a given stage of the merging process. A pair of objects to merge is a solution. All the possible solutions form the solution space, and the possible solutions

which must be checked to find the final solution form the search space. The problem of reducing the dimensions of the search space is as old as Artificial Intelligence [2].

Let us assume that we have n objects, and that among them there are m pairs which can be merged. This means that finally there will be $n-m$ objects, because one merged object repalces its two components. Let, at the step i of the merging process, a pair to be merged be found among all the possible p_i pairs after $t_i = \alpha_i p_i$ trials, where $\forall_i\ \alpha_i \in \langle 0, 1\rangle$. In the first step, we have

$$t_0 = \alpha_0 p_0 = \alpha_0 n(n-1)/2. \tag{1}$$

In each step, the number of objects decreases by one. The number of necessary trials is

$$t_i = \alpha_i p_i = \alpha_i (n-i)(n-i-1)/2. \tag{2}$$

The total number of trials T is

$$T = \sum t_i = \frac{1}{2}\sum_{i=0}^{m} \alpha_i (n-i)(n-i-1). \tag{3}$$

In each case, such an "average" $\overline{\alpha}$ can be found that[1] $T = \frac{1}{2}\sum_{i=0}^{m} \alpha_i (n-i)(n-i-1) = \frac{\overline{\alpha}}{2}\sum_{i=0}^{m}(n-i)(n-i-1)$. Then, Eq. (3) can be rewritten as

$$T = \frac{\overline{\alpha}}{2}\sum_{i=0}^{m}(n-i)(n-i-1). \tag{4}$$

If $\overline{\alpha} = 1.0$, and if a half of the initial segments are merged, then for $m = 100$, it is $T = 147,050$, and for $m = 1000$, it is[2] $T = 145,958,000$.

The smaller $\overline{\alpha}$ is, the smaller fraction of the solution space is searched, and the more efficient the merging algorithm is. In this way, $\overline{\alpha}$ can be used as a measure of efficiency of the merging algorithms. We shall call it the efficiency factor of the merging algorithm.

In the present paper a simple method of reducing the search space in the problem of merging objects is proposed. In each particular case, there exists an optimal order in which pairs of objects should be joined. The presented concept in general does not lead to that order, which depends on the type of merged objects. In such an optimal order, spatial vicinity of objects should be taken into account. In a 2D case this does not impose a linear ordering. A suboptimal order can be found, however, and an important increase in the merging process effectiveness can be obtained.

Due to the suboptimality of the solution found, a case in which no increase or even a decrease of effectiveness can occur, if an initial ordering is very near to optimality. Such a case seems to be very rare in practice.

2. The method

The proposed method will be presented with the simplest example of linear segments. A good and simple precondition for merging two segments is that their inclinations do not

[1] In the most simple algorithm, the last coefficient $\alpha_m = 1$, because all the possible p_m pairs must be checked to prove that there are none to merge. In the presented algorithm this will not be the case.

[2] In the examples shown in Section 3, $\overline{\alpha}$ was between 0.01 and 0.05.

differ too much:

$$\Delta(s_1, s_2) = |\mathrm{Red}(\theta_{s_1} - \theta_{s_2})| < t_\theta, \tag{5}$$

where:

θ_s	–	angle between the segment s and the x axis;
$\mathrm{Red}(\theta)$	–	θ reduced to the interval $\langle 180°, -180° \rangle$;
t_θ	–	threshold for angle θ.

This precondition can be applied irrespective of the form of the final, sufficient condition for merging, which can be much more complex. The final condition can even include the search for the best segment to be merged with a given one.

Linear segments can be sorted according to their angle of inclination θ with reference to a coordinate system[3]. Now, the existence of the ordering makes it possible *not to check* the precondition for all the pairs of segments, as it should be checked *only* for those segments which are near in the ordered list. Let us fix our attention on a segment s_i, and consider forward checking. If for a specified j, $j > i$, it is $\Delta(s_i, s_j) < t_\theta$, and $\Delta(s_i, s_{j+1}) \geq t_\theta$; then, due to the ordering, it is also[4] $\forall_{k \geq 1}\ \Delta(s_i, s_{j+k}) \geq t_\theta$. Hence, once a segment which does not fulfil the precondition (5) is encountered, *no more* segments behind it should be checked. The similar concerns backward checking. This is here where the source of the power of the proposed algorithm lies, which will be clearly demonstrated further by the examples.

It seems reasonable to check one segment backward and one forward, and so on, to examine the smallest differences $\Delta(.,.)$ first. Obviously, no pair should be considered twice, so, except the first steps of the algorithm for a given segment, only forward checking is necessary. Once a new segment is formed of two merged ones, it is pasted in its right place in the list, according to the ordering. The merging process restarts from that new segment; this is the most efficient procedure mainly in the final stage of the process, as the pairs containing the new segment are then the nearly only ones still not checked.

Different than in [12], where merging was considered only for the segments which emerged as immediate neighbours during the segmentation of a curve (cf. [12], Section 7), in the proposed algorithm *all* the pairs of potentially mergeable segments are examined.

The overhead for initial sorting is not large; in Section 3 a comparison of calculation times is given for the example with the largest number of segments (see Tab. 5).

Similarly as in the case of linear segments with their angle of inclination, other features can be used for sorting of various kinds of objects. For circular segments, sorting according to their radius can be used. For elliptic or other quadric segments, a good feature is their offset. The segments of logarithmic spirals, described in the radial coordinate system (ϱ, φ) by an equation $\varrho = ae^{k\varphi}$, can be sorted according to the coefficient k in the exponent.

[3] This angle is a periodic function, so the sorted list of segments is in fact a circular list.

[4] In the case of a circular list, it is in fact $\exists_{k \geq 1}\ \Delta(s_i, s_{j+k}) \geq t_\theta$, which does not change the algorithm, as forward and backward checking is performed.

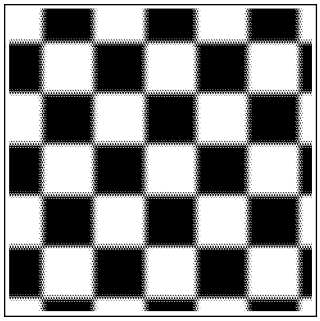


Fig. 1. Example image 1. Window: $236 * 236$ pixels.

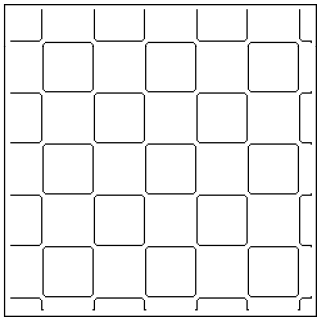


Fig. 2. Improved edges found in the image of Fig. 1.

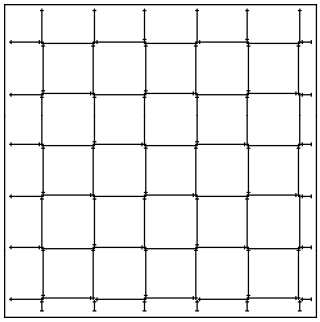


Fig. 3. Segments found in edges of Fig. 2, before merging.

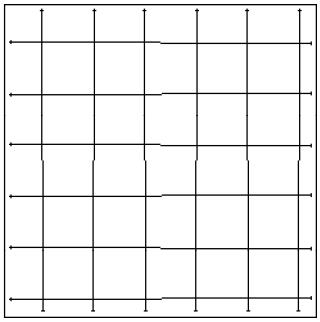


Fig. 4. Segments of Fig. 3, after merging.

Numerous other examples can be given. We put aside the problem of how to reliably estimate the considered feature from the image. If merging is attempted, such a feature should be found, anyway. In the case of linear segments, which is of our main interest, the necessary feature is relatively easy to calculate.

3. Examples

In the four following examples of merging linear segments the threshold in the precondition (5) was $t_\theta = 10°$. The used final condition for merging consisted of four partial conditions.

The first partial condition was the same as the precondition (5). This necessitates for an explanation. It must be stressed that in the case of applying (5) as a part of

algorithm		segments			segment pairs considered			
ord	**prc**	**ini**tial	**merg**ed	**mrg/ini**	total	**prc** OK	$\overline{\alpha}$	saving
NO	NO	84	72	85.7%	1622	-	0.0165	0.0%
YES	NO		72	85.7%	2712	-	0.0276	−67.2%
YES	YES		72	85.7%	1401	1251	0.0127	23.0%

Tab. 1. Comparison of results for example image 1 of Fig. 1, for merging algorithms with and without a precondition (**prc**, Eq. (5)) and an ordering (**ord**). **prc** OK is the no. of pairs which fulfilled the precondition. Saving: according to (8).

algorithm		segments			segment pairs considered			
ord	**prc**	**ini**tial	**merg**ed	**mrg/ini**	total	**prc** OK	$\overline{\alpha}$	saving
NO	NO	294	132	44.9%	79879	-	0.0226	0.0%
YES	NO		119	40.5%	151425	-	0.0451	−99.6%
YES	YES		116	39.4%	26925	25089	0.0076	66.5%

Tab. 2. Comparison of results for example image 2 of Fig. 5. Notations as in Tab. 1.

the final condition, it was *not used* to restrict the search for a match with the current segment to those segments which were its immediate neighbours on the list. This remark is important, as also the version of the merging algorithm *without* sorting this list was considered for comparison.

The second partial condition was similar to that described in [12]. It was based on the notion of the *quality* Q_s of the segment s. Two segments: s_1 and s_2 can be merged to form s_3 if

$$Q_{s_3} \geq \min(Q_{s_1}, Q_{s_2}). \tag{6}$$

The quality measure Q_s was described as

$$Q_s = l_s / E_s, \tag{7}$$

where:

l_s – length of the segment s;

E_s – mean square error of the least squares fit of pixels to the segment s.

The third partial condition was that the distance between the nearest ends of the segments s_1 and s_2 was less than a threshold. The fourth one provided for that one segment could be merged with another one if it were its extension. If a projection of one segment on the direction of the second one had a large common part with that second one, the segments were not merged.

The segments were sought in the image with the *bidirectional expand – contract* method (BECM) described in [13]. The normal-angle representation of segments was used, so θ_s and l_s were readily known. Merging and calculating Q_{s_3} was simplified, as

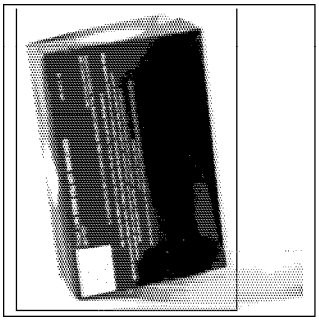


Fig. 5. Example image 2. Window: $236 * 171$ pixels.

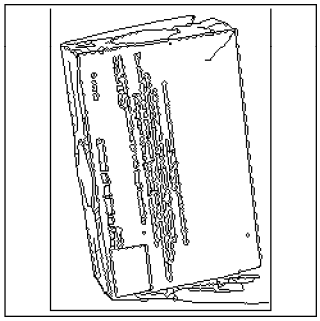


Fig. 6. Improved edges found in the image of Fig. 5.

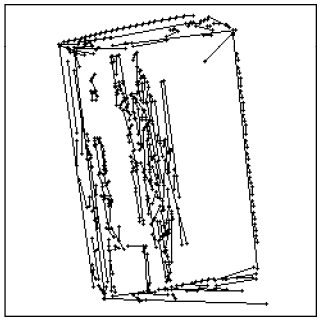
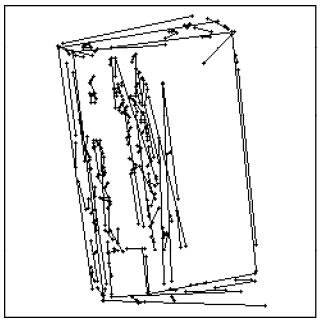
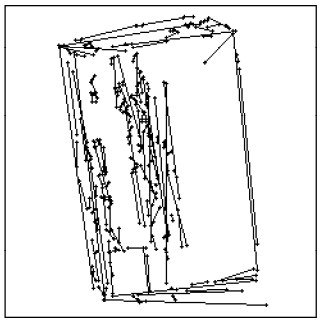


Fig. 7. Segments found in edges of Fig. 6, before merging.





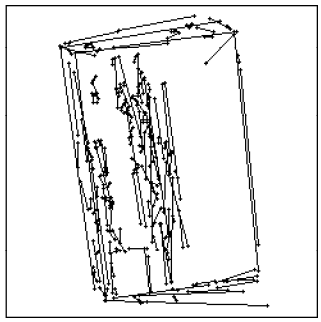


Fig. 8. Segments of Fig. 7 merged with the algorithms: left – no ordering, no precondition; middle – with ordering, but with no precondition; right – with an ordering and a precondition.

in BECM all the necessary statistics of positions of the pixels[5] which form the segment are kept as intermediate results.

To compare the efficiency of the algorithms, a relative measure of the number of saved comparisons S can be introduced:

$$S = 1 - \overline{\alpha}/\overline{\alpha}_r, \tag{8}$$

where $\overline{\alpha}$ and $\overline{\alpha}_r$ are the efficiency factors in the tested and the reference algorithm, respectively[6].

In the reference algorithm, pairs were checked in the order in which they have been found. Two algorithms will be compared with that reference one.

[5] of the form $\sum_{i=1}^{n} x_i^k y_i^l$, where n – number of pixels (x_i, y_i); $k, l = 0, 1, 2$.

[6] If n and m are the same in both algorithms, then $S = (T_r - T)/T_r$, where T, T_r – numbers of segment pairs considered in the algorithms.

algorithm		segments			segment pairs considered			
ord	**prc**	**ini**tial	**merg**ed	**mrg/ini**	total	**prc** OK	$\overline{\alpha}$	saving
NO	NO	483	81	16.8%	424079	-	0.0528	0.0%
YES	NO		76	15.7%	421869	-	0.0553	−4.7%
YES	YES		75	15.5%	41354	38539	0.0051	90.3%

Tab. 3. Comparison of results for example image 3, not shown (very similar to that of Fig. 9). Notations as in Tab. 1.

algorithm		segments			segment pairs considered			
ord	**prc**	**ini**tial	**merg**ed	**mrg/ini**	total	**prc** OK	$\overline{\alpha}$	saving
NO	NO	410	77	18.8%	387805	-	0.0720	0.0%
YES	NO		79	19.3%	405834	-	0.0738	−2.6%
YES	YES		70	17.0%	29028	26832	0.0054	92.5%

Tab. 4. Comparison of results for example image 4 of Fig. 9. Notations as in Tab. 1.

The first one is the algorithm with a sorted list of segments, but without a precondition. The second one is the proposed algorithm with sorting and with a precondition, as described in Section 2[7].

The first example (Figs. 1-4) is an artificial image designed to show the ability of the merging algorithms to find the geometrical relations between segments in a correct way.

The three following examples (Figs. 5-12) have been chosen from the experiments aimed at finding features which could be an input for visual servo control (see e.g. [7]). The experiments were aimed at control of movement of a box-shaped object, and the sought features were the vertices of the largest parallelogram in the image [17]. The algorithm of finding the parallelogram consisted in hypothesizing one on the basis of pairs of long, intersecting segments, and verifying that hypothesis by seeking support for the remaining two edges among the other segments present in the image. The parallelogram with the strongest support or with the largest area was accepted as a result (see [17] for details).

The parallelograms obtained for the images of Fig. 12 are shown in Fig. 13. The results based on the segments obtained with the three considered algorithms do not differ significantly from each other.

Comparison of results obtained for the examples are shown in Tabs. 1-4. It can be seen that the algorithm with sorting the list of merged segments and with a precondition is much more efficient than the reference algorithm. The saving obtained in the presented

[7]The third possible combination, precondition without ordering, has no sense, as the precondition can be applied only to the sorted list of objects to be merged.

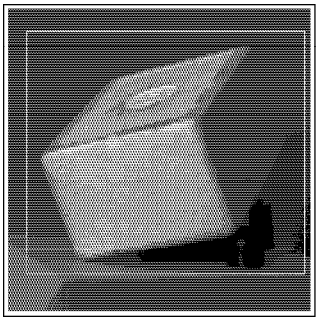


Fig. 9. Example image 4. Window: $189 * 216$ pixels.

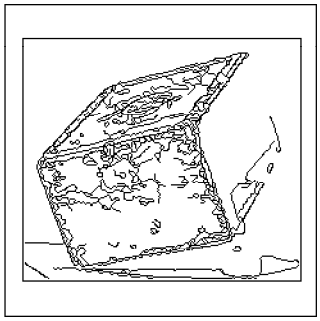


Fig. 10. Improved edges found in the image of Fig. 9.

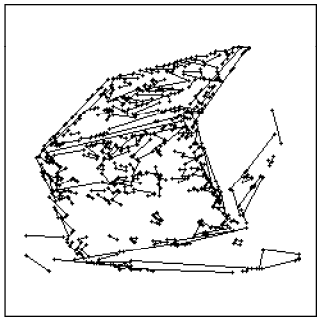
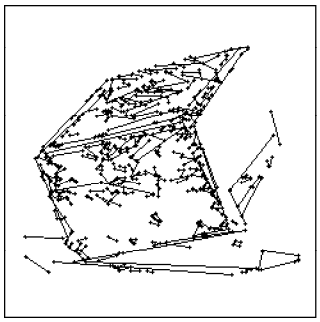
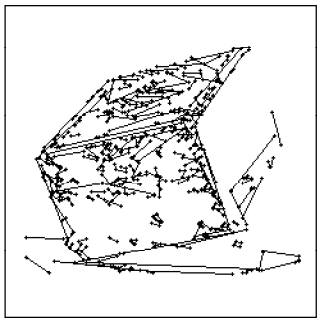


Fig. 11. Segments found in edges of Fig. 10, before merging.





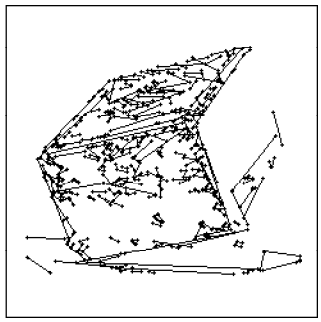


Fig. 12. Segments of Fig. 11 merged with the algorithms: left – no ordering, no precondition; middle – with ordering, but with no precondition; right – with an ordering and a precondition.

examples is between 23 and 93%. The same can not be said of the algorithm with the sorting, but without the precondition. The results indicate that the concept of merely trying to merge the "more parallel" segments first does not work well.

A comparison of times required for the subsequent stages of the merging algorithms for the example shown in Tab. 3 is presented in Tab. 5. The extra time needed for sorting[8] is small in comparison to the saving obtained with the precondition.

4. Conclusion

A general method of reducing the search space in the problem of merging objects of various kinds has been proposed. It consists of using an ordering of objects according

[8] This time depends on the quality of the sorting algorithm, and could be further reduced.

algorithm		time [s]			
ord	**prc**	sorting	merging	total	%
NO	NO	0.0	10.4	10.4	100
YES	NO	1.0	10.1	11.1	107
YES	YES	1.0	2.3	3.3	32

Tab. 5. Times of the stages of the merging algorithms for the example of Tab. 3. Calculations executed on a 80486 AT, 50 MHz.

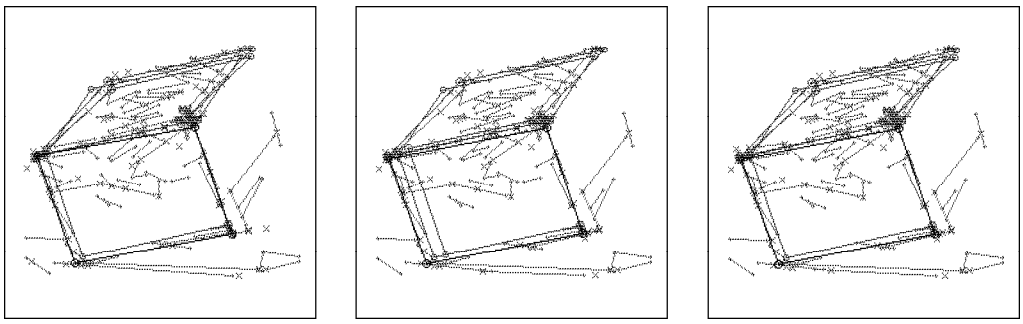


Fig. 13. The largest parallelogram (dark black) found from the segments of Fig. 12, i.e., merged with the algorithms: left – no ordering, no precondition; middle – with ordering, but with no precondition; right – with an ordering and a precondition.

to a chosen feature to preselect the good candidates for merging. If these objects which are close to each other according to the ordering are good candidates for merging, a substantial reduction of the number of pairs of objects which should be checked for mergeability can be achieved. It should be noted, however, that this simple algorithm is suboptimal. Hence, in the worst case, that is, when the initial order of merged objects is very close to an optimal one, a loss of efficiency can be obtained. Such a case has not been encountered in practice.

Examples of merging linear segments were presented. The saving of the number of considered pairs was between about 20 and 90%.

The proposed method has been used to enhance the quality of image features used as an input to a visual servo control algorithm.

Acknowledgement This research has been financed by the Polish Committee for Scientific Research under the grant KBN 8 S505 014 05: *Development of the methods of motion analysis in images, in application to videoservomechanisms.*

References

1982
[1] Ballard D., Brown C.M.: Computer Vision. Prentice Hall, Englewood Cliffs.

1985
[2] Charniak E., McDermott D.: Introduction to Artificial Intelligence. Addison Wesley.

1986
[3] Nalwa V.S., Pauchon E.: Edgel aggregation and edge description. *Proc. 8th ICPR, Paris, Oct. 27-31, 1986*, 604-609.

1987
[4] Illingworth J., Kittler J.: The adaptive Hough transform. *IEEE Trans. PAMI*, 9(5), 690-697.
[5] Li H., Lavin M.A., LeMaster R.J.: Fast Hough transform: A hierarchical approach. *CVGIP*, 36, 139-161.
[6] Pavlidis T.: *Grafika i Przetwarzanie Obrazów*. Biblioteka Inżynierii Oprogramowania. WNT. (*Algorithms for Graphics and Image Processing*, Computer Science Press Inc., Rockville, 1982).
[7] Weiss L.E., Sanderson A.C., Neuman C.P.: Dynamic sensor-based control for robotics with visual feedback. *IEEE J. RA*, 3(5), 404-416.

1988
[8] Illingworth J., Kittler J.: A survey of the Hough transform. *CVGIP*, 44, 87-116.

1989
[9] Dolan J., Weiss R.: Perceptual grouping of curved lines. In *Proc. DARPA Image Understanding Workshop, Palo Alto, CA, May 23-26, 1989*, 1135-1145.
[10] Silbermann M.J., Sklansky J.: Toward line detection by cluster analysis. *Proc. 3rd CAIP, Leipzig, Sept. 8-10, 1989*, 117-121.

1990
[11] Dambra C., Serpico S.B., Vernazza G.: A new technique for peak detection in the Hough-transform parameter space. *Proc. 5th European Signal Processing Conf., Barcelona, Sept. 18-22, 1994*, 705-708.

1991
[12] West G.A.W., Rosin P.L.: Techniques for segmenting image curves into meaningful descriptions. *Pattern Recognition*, 24(7), 643-652.

1992
[13] Chmielewski L.: Instead of the Hough transform - Part one: Sliding estimate of line segments. *MG&V*, 1(1-2), 269-286.
[14] Słówko M.: Instead of the Hough transform - Part two: Finding geometrical relations between line segments. *MG&V*, 1(3), 537-542.

1993
[15] Huddleston J.N., Ben-Arie J.: Grouping edgels into structural entities using circular symmetry, the Distributed Hough Transform, and probabilistic non-accidentalness. *CVGIP: Image Understanding*, 57(2), 227-242.

1994
[16] Foresti G., Murino V., Regazzoni C.S., Vernazza G.: Grouping of rectilinear segments by the Labelled Hough Transform. *CVGIP: Image Understanding*, 58(3), 22-42.

1995
[17] Chmielewski L.: Selected low and middle level computer vision programs for extracting the features from an image, for control of relative movement of the camera and objects in its field of view (in Polish). Report within the research project KBN no. 8 S505 014 05 "Development of the methods of motion analysis in images, in application to videoservomechanisms", EPSILON ARG, January.