On the Calculation of Image Moments

Jan Flusser and Tomáš Suk

Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 4, 182 08 Prague 8, Czech Republic
E-mail: {flusser,suk}@utia.cas.cz

RESEARCH REPORT No. 1946, JANUARY 1999

Abstract

Numerous methods for effective calculation of image moments have been presented up to now. In this paper, we present a new one which is particularly effective when one wants to calculate moments of more than one image. The major idea is that in the new method a part of calculations does not depend on the object and thus it can be performed only once in advance. The object-dependent part is then realized very quickly. The method works for arbitrary binary 2-D shapes and can be easily extended to n-D case. The MATLAB codes accompany the description of the algorithms.

Keywords: Binary image, discrete image moments, object boundary, effective calculation, n-D moments

---

This work has been supported by the grants No. 102/96/1694 and 102/98/P069 of the Grant Agency of the Czech Republic.
In a shorter form, this work was submitted to the journal *Pattern Recognition Letters*.

1. Introduction

Image moments and various types of moment-based invariants play very important role in object recognition and shape analysis [1], [2], [3], [4]. The $(p+q)$th order geometric moment M_{pq} of a grey-level image $f(x,y)$ is defined as

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y) dx dy. \tag{1}$$

In the case of a digital image, the double integral in eq. (1) must be replaced by a summation. The most common way how to do that is to employ the rectangular (i.e. zero-order) method of numeric integration. Then (1) turns to the well-known form

$$m_{pq} = \sum_{i=1}^{N} \sum_{j=1}^{N} i^p j^q f_{ij}, \tag{2}$$

where N is the size of the image and f_{ij} are the grey levels of individual pixels.

However, m_{pq} is just an approximation of M_{pq}. As it was pointed out by Lin [5], more precise approximation can be obtained by exact integration of the monomials $x^p y^q$:

$$\hat{m_{pq}} = \sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} \int \int_{A_{ij}} x^p y^q dx dy =$$

$$= \frac{1}{(p+1)(q+1)} \sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} ((i+\frac{1}{2})^{p+1} - (i-\frac{1}{2})^{p+1})((j+\frac{1}{2})^{q+1} - (j-\frac{1}{2})^{q+1}), \tag{3}$$

where A_{ij} denotes the area of the pixel (i,j).

Since direct calculation of discrete moments by (2) or (3) is time-consuming (it requires $O(pqN^2)$ operations), a large amount of effort has been spent in the last decade to develop more effective algorithms [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. Particular attention has been paid to binary objects because of their importance in practical pattern recognition applications. Since any binary object is fully determined by its boundary, various boundary-based methods of moment calculation have been developed.

The first attempt to speed up the moment calculation came from Zakaria [6]. The basic idea of his "Delta" method is to decompose the object to the individual rows of pixels. The object moment is then given as a sum of all row moments, which can be easily calculated just from the coordinates of the first and last pixels. Zakaria's method worked for convex shapes only and dealt with moment approximation (2). Dai [7] extended Zakaria's method also to approximation (3) and Li [8] generalized it for non-convex shapes. Recently, Spiliotis and Mertzios [16] have published an advanced modification of Delta method. Their algorithm employs block-wise object representation instead of the row-wise one. Thanks to this, it works faster than the original version.

Another group of methods is based on Green's theorem, which evaluates the double integral over the object by means of single integration along the object boundary. Li and Shen [12] proposed a method based on Green's theorem in continuous domain. However, their results depend on the choice of the discrete approximation of the boundary and differ from the theoretical values. Jiang and Bunke [11] approximated the object by a polygon first and then they applied the Green's theorem. Thanks to this, they calculated only single integrals along line segments. Unfortunately, due to two-stage approximation, their method produce inaccurate results. Philips [15] proposed to use discrete Green's theorem instead of the continuous one. For convex shapes, his approach leads to the same formulae as the Delta method and it was shown to yield exact moment values. Recently, Yang and Albregtsen [13] and Sossa [14] have slightly improved the speed of Philips' method.

There were also described methods based on polygonal approximation of the object boundary. Object moments are then calculated via the corner points [9], [10]. These methods are efficient only for simple shapes with few corners. Another approach published in [17] and [18] shows that moment computation can be effectively implemented in parallel processors. Chen [17] proposed a recursive algorithm for a SIMD processor array, Chung [18] presented a constant-time algorithm on reconfigurable meshes.

In this paper, we present a new method for moment calculation in the case of a sequence of objects. All methods published up to now have considered single object only. However, in practical shape recognition tasks we have to deal with a large number of objects. Since the same set of moments is to be calculated for each object, it would be highly desirable to pre-calculate some operations which are common for all objects in advance. Thus, we modify the Philips' method in the following way: we divide the algorithm in two parts – in a common one, which is performed only once at the beginning, and in a part which is performed for each individual object. To make the both parts as efficient as possible, we describe the algorithms in terms of matrix algebra, which allows an effective implementation in MATLAB. Moreover, we propose a faster boundary detection method than Philips has published, which does not require any contour tracing. We present two versions of the new method which utilise moment approximation (2) and (3), respectively. All algorithms described in this paper are accompanied by MATLAB codes and their performance is illustrated by numerical experiments and by a comparison with standard moment calculation techniques. At the end of the paper we present also an effective MATLAB code for calculation of moments of grey-level images and an extension of the method to computing n-dimensional moments.

2. Calculating moments from object boundary

In this Section, we present a fast algorithm for computing moments of binary object with pre-calculations. We adopt the Philips' definition of the left-hand side and right-hand side boundaries $\partial\Omega-$ and $\partial\Omega+$, respectively.

Let Ω be a bounded subset of a discrete plane. Then we define

$$\partial\Omega- = \{(x,y)|(x,y) \notin \Omega, (x+1,y) \in \Omega\},$$
$$\partial\Omega+ = \{(x,y)|(x,y) \in \Omega, (x+1,y) \notin \Omega\}.$$

Furthermore, we denote the union of the left and right boundaries as $\partial\Omega\pm$. Note that $\partial\Omega\pm$ differs from the "normal" boundary of Ω in many ways: for instance, $\partial\Omega- \not\subset \Omega, \partial\Omega+ \subset \Omega, \partial\Omega\pm$ is not a closed curve, etc.

Decomposing Ω into row segments we express the formulae for object moments (2) and (3) by means of $\partial\Omega-$ and $\partial\Omega+$:

$$m_{pq} = \sum_{\partial\Omega+} y^q \sum_{i=1}^{x} i^p - \sum_{\partial\Omega-} y^q \sum_{i=1}^{x} i^p, \tag{4}$$

$$\hat{m_{pq}} = \frac{1}{(p+1)(q+1)}[\sum_{\partial\Omega+}(x+\frac{1}{2})^{p+1}((y+\frac{1}{2})^{q+1}-(y-\frac{1}{2})^{q+1})-\sum_{\partial\Omega-}(x+\frac{1}{2})^{p+1}((y+\frac{1}{2})^{q+1}-(y-\frac{1}{2})^{q+1})]. \tag{5}$$

Proof of eq. (4):

Let's consider a horizontal line segment L of the lenght n

$$L = \{(x_1, y), (x_2, y), \cdots, (x_n, y)\}.$$

Both boundaries of L consist of one pixel only:

$$\partial L- = \{(x_1 - 1, y)\},$$

$$\partial L+ = \{(x_n, y)\}.$$

Discrete moments of L are given by (2) as

$$m_{pq}^{(L)} = y^q \sum_{i=x_1}^{x_n} i^p = y^q (\sum_{i=1}^{x_n} i^p - \sum_{i=1}^{x_1 - 1} i^p).$$

Thus, eq. (4) holds for line segments.

Let's consider an object Ω with the boundaries

$$\partial\Omega- = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\},$$

$$\partial\Omega+ = \{(x'_1, y_1), (x'_2, y_2), \cdots, (x'_m, y_m)\}.$$

This object can be decomposed into m line segments L_j:

$$\Omega = \bigcup_{j=1}^{m} L_j,$$

where

$$L_j = \{(x_j + 1, y_j), (x_j + 2, y_j), \cdots, (x'_j, y_j)\}.$$

The object boundaries can be decomposed too:

$$\partial\Omega- = \bigcup_{j=1}^{m} \partial L_j -,$$

$$\partial\Omega+ = \bigcup_{j=1}^{m} \partial L_j + .$$

The moment of Ω is given as a sum of moments of the segments L_j. Thus, it holds

$$m_{pq}^{(\Omega)} = \sum_{j=1}^{m} m_{pq}^{(L_j)} = \sum_{j=1}^{m} y_j^q (\sum_{i=1}^{x'_j} i^p - \sum_{i=1}^{x_j} i^p) = \sum_{\partial\Omega+} y^q \sum_{i=1}^{x} i^p - \sum_{\partial\Omega-} y^q \sum_{i=1}^{x} i^p. \qquad \square$$

The proof of eq. (5) is quite similar.

Note that Philips [15] derived the formula (4) too but in more complicated way using the discrete Green's theorem.

The above formulae can be advantageously expressed in matrix forms. Let R and S be $p_m \times N$ matrices ($p_m \geq p+1, p_m \geq q+1$) defined as follows:

$$R_{ij} = j^{i-1},$$

$$S_{ij} = \sum_{n=1}^{j} n^{i-1}.$$

Now (4) turns to the form

$$m_{pq} = \sum_{\partial\Omega+} S_{p+1,x} \cdot R_{q+1,y} - \sum_{\partial\Omega-} S_{p+1,x} \cdot R_{q+1,y}. \tag{6}$$

Let P be a $p_m \times (N+1)$ matrix defined as

$$P_{ij} = \frac{1}{i}(j - \frac{1}{2})^i.$$

Eq. (5) becomes now

$$\hat{m_{pq}} = \sum_{\partial\Omega+} P_{p+1,x+1} \cdot (P_{q+1,y+1} - P_{q+1,y}) - \sum_{\partial\Omega-} P_{p+1,x+1} \cdot (P_{q+1,y+1} - P_{q+1,y}). \tag{7}$$

The major idea of our new method is based on the fact that the matrices R, S and P don't depend on the object Ω, which is currently under investigation. Thus, they can be pre-calculated only once at the beginning. Their size must be appropriately chosen according to the expected size of the object and to the highest order of moments we want to calculate.

The boundaries $\partial\Omega-$ and $\partial\Omega+$ are the only things in eqs. (6) and (7) depending on the object. To find them, Philips proposed a contour tracing algorithm. We propose a faster detection by convolving the image matrix and the vector $h = (0, 1, -1)$. After that the pixels having the value -1 and 1 correspond to those belonging to $\partial\Omega-$ and $\partial\Omega+$, respectively (assuming that Ω is represented by ones on a zero background).

This method works for objects of any shape, even for object with holes and with several disconnected components.

3. Complexity analysis

In this Section, we give an analysis of the computing complexity of the both steps of the method.

Generating each of the matrices R, P and S requires only $O(N \cdot p_m)$ operations. Thanks to the effective implementation, the actual computing times are reasonably low (see Table 1). It should be noted that the complexity of this stage does not affect the overall complexity significantly, because this stage is performed only once. If the number of objects under consideration is high, the complexity of the first stage becomes negligible.

p_m	128	256	512
3	10	60	270
5	10	70	330
7	10	110	380
10	40	120	440
20	50	160	770

Table 1: The times (in milliseconds) needed for generating matrices R and S together depending on their sizes. The experiment was performed on PC Pentium 200 MHz.

The moment computation itself (stage two) is very fast. It requires only K multiplications and $(K-1)$ additions in the case of formula (6) (i.e. two elementary operations per each boundary pixel) or K multiplications and $(2K-1)$ additions (i.e. three elementary operations per each boundary pixel) when we use the formula (7). K denotes the number of pixels belonging to $\partial\Omega\pm$. Note that the complexity does not depend on the moment order at all. This means any moment whose indices are less than p_m can be calculated as quickly as m_{00}.

Analyzing the complexity of previously published methods, one can see that our method outperforms all of them. For more details we refer to [13] where a survey of the complexity of the recent algorithms is given.

4. Implementation using MATLAB

In this Section, we present an implementation of the above described method in programming language MATLAB 5.1. Since MATLAB provides very effective tools for implementing matrix operations, it is quite convenient for programming almost all image processing algorithms including moment calculation. The MATLAB codes of two procedures are presented below. The first one calculates the matrices R and S, the second one performs the boundary detection and moment calculation for a given object using the formula (6).

For implementation purposes it is more convenient to generate matrices R and S as the results of elementary vector and matrix operations rather than to calculate them directly from their definitions. The matrix R can be generated as

$$R = \left(\begin{pmatrix} 1 \\ 1 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{pmatrix} \cdot (1, 2, 3, \cdots N) \right) \wedge \left(\begin{pmatrix} 0 \\ 1 \\ 2 \\ \cdot \\ \cdot \\ \cdot \\ p_m - 1 \end{pmatrix} \cdot (1, 1, 1, \cdots 1) \right) \tag{8}$$

where $\wedge$ denotes involution elementwise. Matrix S is then computed as a product of R and an

$N \times N$ upper triangular matrix T, $T_{ij} = 1$ for $i \leq j$:

$$S = R \cdot T.$$

```
function [R,S]= gen_rs(N,r)

% Generates the matrices R and S for fast moment calculation
%
% N - expected image size
% r - maximum index of moments we want to calculate

pm=r+1;
R1=ones(pm,1)*linspace(1,N,N);
R2=linspace(0,r,pm)'*ones(1,N);    % Generating subsidiary matrices
R=R1.^R2;                          % Generating R
S=R*triu(ones(N));                 % Generating S
```

```
function m=moment(p,q,B,R,S)

% Fast calculation of geometric moment m_{pq}
%
% B -   binary image matrix
% R,S - previously generated matrices by
%       [R,S]=gen_rs(size(B,1),r), where r >= p and r >= q


d=filter2([1,-1],B);                        % Boundary detection
[i,j]=find(d==-1);                          % List of pixels belonging to dO-
[k,l]=find(d==1);                           % List of pixels belonging to dO+
m=S(p+1,l)*R(q+1,k)'-S(p+1,j)*R(q+1,i)';   % Moment calculation
```

If we want to use the equation (7) instead of (6) for moment calculation, the above procedures can be easily modified.

5. Numerical experiment

The aim of this experiment is to demonstrate that our method gives exactly the same results as a direct evaluation of the equations (2) and (3).

Figure 1: The test object used in the experiment

In Fig. 1, one can see the test object of the size 128×128 pixels. In Table 2 there are summarized the values of some low-order moments obtained by four different algorithms. It can be seen that our method produces exact results in all cases. One can also check the difference between moment values calculated by two different discrete approximation of the original moment definition (1). All experiments presented in this paper were carried out on 200 MHz Pentium PC.

	Eq. (2)	Our method	Eq. (3)	Our method
m_{00}	7650	7650	7650	7650
m_{10}	475 021	475 021	475 021	475 021
m_{20}	36 972 445	36 972 445	36 973 083	36 973 083
m_{11}	24 016 880	24 016 880	24 016 880	24 016 880
m_{30}	$3.3091 \cdot 10^9$	$3.3091 \cdot 10^9$	$3.3092 \cdot 10^9$	$3.3092 \cdot 10^9$
m_{21}	$1.7277 \cdot 10^9$	$1.7277 \cdot 10^9$	$1.7278 \cdot 10^9$	$1.7278 \cdot 10^9$

Table 2: Moments of the object in Fig. 1 obtained by four different methods. From left to right: direct evaluation of eq. (2), our method using eq. (6), direct evaluation of eq. (3) and our method using eq. (7).

6. Effective calculation of moments of grey-level images

To calculate moments of a grey-level image, it is in principle impossible to achieve lower complexity than $O(N^2)$. Regardless of the method used, we have to look at each individual pixel of the image.

Due to this we cannot use any boundary-based method. Fu [19] proposed a moment calculation via Hadamard transform, Shen [20] calculated the image moments from the discrete Radon transform projections. Li [21] introduced a concept of auxiliary moments which correspond to the inner product of the image and a linear combination of monomials. Image moments are then computed by another linear transform. Recently, Martinez et al. further developed this approach in [22] and [23]. We can, however, speed up the computation also by some kind of approximation of the image function and by computing moments of the approximated image. As a rule, this leads to inaccurate results.

In MATLAB we can speed up the enumeration of formula (2) significantly by rewriting it in a matrix form and using the pre-calculated matrix R (8) . The code of the effective algorithm is as follows.

```
function m=moment(p,q,G,R)

% Fast calculation of geometric moment m_{pq}
%
% G - gray-level image matrix
% R - previously generated matrix

m = R(p+1,:)*G*R(q+1,:)';
```

To show the efficiency of the matrix-based implementation, we compare it with a "straightforward" implementation of eq. (2) using double FOR loop. The results achieved for images of various sizes and for various moments show that the matrix method is more than 100 times faster. For instance, the calculation of m_{30} of the 512×512 Lena image by straightforward implementation took 9.7 seconds and by the matrix method it took only 0.05 seconds. For a comparison, Li wrote in [21] his method needs 0.22 seconds to calculate third-order moments of a 128×128 image on a VAX/785 computer.

7. Extension to n-dimensional objects

During last few years, the necessity of dealing with digital images of more than two dimensions have appeared, particularly in medicine in NMR and CT imaging. In order to describe or recognize n-D shapes by means of moment invariants, one has to calculate n-D object moments. Since the straightforward calculation requires $O(N^n)$ operations (N is the size of a hypercube containing the object), effective algorithms become even more important than in 2-D case.

In comparison with 2-D moment computation, there are only few papers on effective calculation of high-dimensional moments. All of them are generalizations of authors' previous works in two dimensions. Li [24] proposed a method for objects represented by a polyhedra, Li and Ma [25] suggested to employ appropriate linear transformations of moments to reduce the computing complexity and Yang et al. [26] developed a method of the moment calculation from the object boundary via discrete divergence theorem. In this Section, we generalize the method from Section 2 to n-dimensional objects.

The moment of n-D image f is defimed as

$$M_{p_1 \cdots p_n} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} x_1^{p_1} x_2^{p_2} \cdots x_n^{p_n} f(x_1, x_2, \cdots, x_n) dx_1 dx_2 \cdots dx_n \tag{9}$$

and the corresponding discrete approximation is

$$m_{p_1 \cdots p_n} = \sum_{i_1=1}^{N} \sum_{i_2=1}^{N} \cdots \sum_{i_n=1}^{N} i_1^{p_1} i_2^{p_2} \cdots i_n^{p_n} f_{i_1 \cdots i_n}. \tag{10}$$

Let Ω be a bounded subset of a discrete n-dimensional space. Then we define its left and right boundaries in accordance with those from Section 2:

$$\partial\Omega- = \{(x_1, x_2, \cdots, x_n) | (x_1, x_2 \cdots, x_n) \notin \Omega, (x_1 + 1, x_2 \cdots, x_n) \in \Omega\},$$

$$\partial\Omega+ = \{(x_1, x_2 \cdots, x_n) | (x_1, x_2 \cdots, x_n) \in \Omega, (x_1 + 1, x_2 \cdots, x_n) \notin \Omega\}.$$

The analogon of eq. (4) in n dimensions is

$$m_{p_1 \cdots p_n} = \sum_{\partial\Omega+} x_2^{p_2} \cdots x_n^{p_n} \sum_{i=1}^{x_1} i^{p_1} - \sum_{\partial\Omega-} x_2^{p_2} \cdots x_n^{p_n} \sum_{i=1}^{x_1} i^{p_1}. \tag{11}$$

Using matrix notation, (11) turns to the form

$$m_{p_1 \cdots p_n} = \sum_{\partial\Omega+} S_{p_1+1, x_1} \cdot \prod_{i=2}^{n} R_{p_i+1, x_i} - \sum_{\partial\Omega-} S_{p_1+1, x_1} \cdot \prod_{i=2}^{n} R_{p_i+1, x_i}. \tag{12}$$

where matrices R and S are the same as those in Section 2.

By means of (12) we reduced the number of operations to n per each boundary voxel.

8. Summary and conclusion

In this paper, we have presented a new method for calculating moments of binary objects. The major advantage of the method is that it decomposes the moment calculation into two stages. First stage, containing relatively complex operations, is performed only once and its results can be used for any object. The second stage contains very simple operations and must be performed for each individual object. This decomposition makes the method extremely effective for extensive object recognition tasks with hundreds of objects under investigation. Both stages of the method were effectively described by matrix algebra and implemented in MATLAB 5.1.

Our method can be used for objects of any shape, even for objects with holes and for those consisting of disconnected components. The method has been proven to give exact values of the moments. The computing complexity of the method is very low: for any object it requires only 2 operations per boundary pixel (it does not include the common pre-calculations and the boundary detection), that clearly outperforms all previously published methods. It was also shown that the method can be easily generalized to calculate moments in higher dimensional space.

References

[1] M. K. Hu, "Visual pattern recognition by moment invariants," *IRE Trans. Information Theory*, vol. 8, pp. 179–187, 1962.

[2] S. O. Belkasim, M. Shridhar, and M. Ahmadi, "Pattern recognition with moment invariants: a comparative study and new results," *Pattern Recognition*, vol. 24, pp. 1117–1138, 1991.

[3] J. Flusser, T. Suk, and S. Saic, "Recognition of blurred images by the method of moments," *IEEE Trans. Image Processing*, vol. 5, pp. 533–538, 1996.

[4] J. Flusser and T. Suk, "Pattern recognition by affine moment invariants," *Pattern Recognition*, vol. 26, pp. 167–174, 1993.

[5] W. G. Lin and S. Wang, "A note on the calculation of moments," *Pattern Recognition Letters*, vol. 15, pp. 1065–1070, 1994.

[6] M. F. Zakaria, L. J. Vroomen, P. Zsombor-Murray, and J. M. van Kessel, "Fast algorithm for the computation of moment invariants," *Pattern Recognition*, vol. 20, pp. 639–643, 1987.

[7] M. Dai, P. Baylou, and M. Najim, "An efficient algorithm for computation of shape moments from run-length codes or chain codes," *Pattern Recognition*, vol. 25, pp. 1119–1128, 1992.

[8] B. C. Li, "A new computation of geometric moments," *Pattern Recognition*, vol. 26, pp. 109–113, 1993.

[9] J. G. Leu, "Computing a shape's moments from its boundary," *Pattern Recognition*, vol. 24, pp. 949–957, 1991.

[10] M. H. Singer, "A general approach to moment calculation for polygons and line segments," *Pattern Recognition*, vol. 26, pp. 1019–1028, 1993.

[11] X. Y. Jiang and H. Bunke, "Simple and fast computation of moments," *Pattern Recognition*, vol. 24, pp. 801–806, 1991.

[12] B. C. Li and J. Shen, "Fast computation of moment invariants," *Pattern Recognition*, vol. 24, pp. 807–813, 1991.

[13] L. Yang and F. Albregtsen, "Fast and exact computation of cartesian geometric moments using discrete Green's theorem," *Pattern Recognition*, vol. 29, pp. 1061–1073, 1996.

[14] J. H. Sossa-Azuela and I. Mazaira-Morales, "An extension to Philip's algorithm for moment calculation," *Pattern Recognition*, submitted in 1997.

[15] W. Philips, "A new fast algorithm for moment computation," *Pattern Recognition*, vol. 26, pp. 1619–1621, 1993.

[16] I. M. Spiliotis and B. G. Mertzios, "Real-time computation of two-dimensional moments on binary images using image block representation," *IEEE Trans. Image processing*, vol. 7, pp. 1609–1615, 1998.

[17] K. Chen, "Efficient parallel algorithms for the computation of two-dimensional image moments," *Pattern Recognition*, vol. 23, pp. 109–119, 1990.

[18] K. L. Chung, "Computing horizontal/vertical convex shape's moments on reconfigurable meshes," *Pattern Recognition*, vol. 29, pp. 1713–1717, 1996.

[19] C. W. Fu and S. Chang, "Calculation of moment invariants via Hadamard transform," *Pattern Recognition*, vol. 26, pp. 287–294, 1993.

[20] T. W. Shen, D. P. K. Lun, and W. C. Siu, "On the efficient computation of 2-D image moments using the discrete Radon transform," *Pattern Recognition*, vol. 31, pp. 115–120, 1998.

[21] B. C. Li, "High-order moment computation of gray-level images," *IEEE Trans. Image Proc.*, vol. 4, pp. 502–505, 1995.

[22] J. Martinez and F. Thomas, "A reformulation of gray-level image geometric moment computation for real-time applications," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Minneapolis, 1996.

[23] J. Martinez, E. Staffetti, and F. Thomas, "A recursive updating rule for efficient computation of linear moments in sliding-window applications," *Proc. Int. Conf. Pattern recognition ICPR'96*, pp. 295–299, Vienna, 1996.

[24] B. C. Li, "The moment calculation of polyhedra," *Pattern Recognition*, vol. 26, pp. 1229–1233, 1993.

[25] B. C. Li and S. D. Ma, "Efficient computation of 3D moments," *Proc. 12th Int. Conf. Pattern Recognition*, vol. I, pp. 22–26, Jerusalem, 1994.

[26] L. Yang, F. Albregtsen, and T. Taxt, "Fast computation of 3-D geometric moments using a discrete divergence theorem and a generalization to higher dimensions," *Graphical Models and Image Processing*, vol. 59, pp. 97–108, 1997.