Available online at www.sciencedirect.com
SCIENCE DIRECT
Image and Vision Computing 24 (2006) 960–969
image AND vision COMPUTING
www.elsevier.com/locate/imavis

Numerical error analysis in Zernike moments computation

G.A. Papakostas [a,*], Y.S. Boutalis [a], C.N. Papaodysseus [b], D.K. Fragoulis [b]

[a] Department of Electrical and Computer Engineering, Democritus University of Thrace, Vas. Sofias 12, 67100 Xanthi, Greece
[b] School of Electrical and Computer Engineering, National Technical University of Athens, 15773 Athens, Greece

Received 4 October 2005; received in revised form 31 January 2006; accepted 16 February 2006

Abstract

An exact analysis of the numerical errors being generated during the computation of the Zernike moments, by using the well-known 'q-recursive' method, is attempted in this paper. Overflow is one kind of error, which may occur when one needs to calculate the Zernike moments up to a high order. Moreover, by applying a novel methodology it is shown that there are specific formulas, which generate and propagate 'finite precision error'. This finite precision error is accumulated during execution of the algorithm, and it finally 'destroys' the algorithm, in the sense that eventually makes its results totally unreliable.

The knowledge of the exact computation errors and the way that they are generated and propagated is a fundamental step for developing more robust error-free recursive algorithms, for the computation of Zernike moments.

Keywords: Zernike moments; Recursive computation; Finite precision error; Numerical stability; Image vision; Feature extraction

1. Introduction

Image moments have been used in image processing applications through the years, since their first introduction by Hu [1]. Their main contribution in the engineering science is their usage as object descriptors, for image coding and pattern classification tasks. *Geometrical, central* and *normalized* moments were for many decades the only family of applied moments. The main disadvantage of these descriptors was their disability to fully describe an object in a way that, by using the moments set, the reconstruction of the object could be possible. In other words, they were not orthogonal.

Orthogonal moments come to fill this gap. Their ability to describe an image fully, with minimum information redundancy, due to their orthogonality property, as well as their robustness in noisy environments, has established them as the most efficient among the moment descriptors.

The most widely used family of orthogonal moments is the Zernike moments, firstly introduced in image processing by Teague [2]. Zernike moments are a set of orthogonal moments that have complex kernel functions based on Zernike polynomials. They have been successfully used, as a tool in image processing and pattern recognition [3–6], because of their ability to decompose an image into its high and low resolution components. Increasing the order of computed Zernike polynomials theoretically to infinity, full image representation can be achieved.

However, there are some computational errors for which one has to be aware, while trying to calculate the Zernike moments. Since these moments are defined as double continuous integral over the domain of normalized coordinates, as it will be presented in Section 2, errors from the discrete approximation of the continuous integrals and of the transformation of the image coordinate system into the domain of the orthogonal polynomials [7], will be involved during the computation. These disadvantages of the continuous orthogonal moments, led to the recently introduced discrete orthogonal moments [7], which are defined directly to the discrete domain.

Additionally, it is proved that there is an inherent limitation in the precision of computing the Zernike moments due to the geometric nature of a circular domain [8]. This error appears owing to the fact that the area used for the moment computation is not equal to that of the unit disk, since some pixels (those which their centers fall outside the unit disk) are not entirely inside the circle and on the other hand, some parts of the circle are not covered by pixels.

* Corresponding author. Tel.: +30 2541079559.
E-mail addresses: gpapakos@ee.duth.gr (G.A. Papakostas), ybout@ee.duth.gr (Y.S. Boutalis), cpapaod@cs.ntua.g (C.N. Papaodysseus), dfrag@mail.ntua.gr (D.K. Fragoulis).

0262-8856/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.imavis.2006.02.015

The above errors that tend to corrupt the accuracy of the computed Zernike moments are referred to the case of computing the moments by using the Zernike polynomials through their definition formula. According to this formula, as it will be discussed in Section 2, there are a lot of computations (factorials) that should be carried out. For this reason, many researchers have introduced methods for fast computation of Zernike polynomials, by computing them recursively [9].

During the last years, a significant research of the error generation and propagation in some signal processing iterative algorithms, with interesting conclusions has been done [10–15]. These approaches have led to the conclusion that there are formulas, which may involve a number of fundamental operations, additions, multiplications, divisions, and subtractions that in an arbitrary iteration of the algorithm, generate finite precision error and some other formulas, propagate this error along the algorithm execution.

Recently, a discussion about the appearance of numerical errors in computing moments by using recurrence equations has been made by Mukundan [16]. However, no research results have been reported on the numerical behaviour of the recursive equations recently used for fast computation of the Zernike moments.

In the present paper, an analysis of the finite precision error, during Zernike moments computation, is attempted along the lines introduced in [9]. More specifically, the authors decided to study the numerical behaviour of the well-known *q-recursive method* [9] as it considered to be the most efficient among the recursive methods used for the Zernike moments computation.

2. Zernike moments

As it has already been mentioned, Zernike moments are the most widely used family of orthogonal moments due to their extra property of being invariant to an arbitrary rotation of the object that they describe. They are used, after making them invariant to scale and translation, as object descriptors in pattern recognition applications [3–5,17–19] and in image retrieval tasks [20,21] with considerable results. Recently, due to their popularity, many methods for accelerating their computation time have been proposed [9,22–26] in order to make them more useful, especially in real time applications, where time is a critical factor.

In the following, the direct computation of Zernike moments that uses the definition of the radial Zernike polynomials is presented, and the computational difficulties for this procedure are discussed.

2.1. Direct computation

The introduction of Zernike moments in image analysis was made by Teague [2], using a set of complex polynomials, which form a complete orthogonal set over the interior of the unit circle $x^2+y^2=1$. These polynomials [3,4] have the form

$$V_{nm}(x,y) = V_{nm}(\rho,\theta) = R_{nm}(\rho)\exp(jm\theta) \tag{1}$$

where n is a non-negative integer and m positive and negative integers subject to the constraints $n-|m|$ even and $|m|\le n$, ρ is the length of vector from the origin $(\bar{x},\bar{y})$ to the pixel(x,y) and θ the angle between vector ρ and x axis in counter-clockwise direction. $R_{nm}(\rho)$, are the Zernike radial polynomials [27] in (ρ,θ) polar coordinates defined as:

$$R_{nm}(\rho) = \sum_{s=0}^{(n-|m|)/2} (-1)^s \cdot \frac{(n-s)!}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n-|m|}{2}-s\right)!}\rho^{n-2s}. \tag{2}$$

Note that $R_{n,-m}(\rho)=R_{nm}(\rho)$.

The polynomials of Eq. (2) are orthogonal and satisfy the orthogonality principle

$$\iint_{x^2+y^2\le 1} V^*_{nm}(x,y)\cdot V_{pq}(x,y)\mathrm{d}x\,\mathrm{d}y = \frac{\pi}{n+1}\delta_{np}\delta_{mq} \tag{3}$$

where $\delta_{\alpha\beta}=1$ for $\alpha=\beta$ and $\delta_{\alpha\beta}=0$ otherwise, is the Kronecker symbol.

The Zernike moment of order n with repetition m for a continuous image function $f(x,y)$, that vanishes outside the unit disk is

$$Z_{nm} = \frac{n+1}{\pi}\iint_{x^2+y^2\le 1} f(x,y)V^*_{nm}(\rho,\theta)\mathrm{d}x\,\mathrm{d}y \tag{4}$$

For a digital image, the integrals are replaced by summations [3,4] to get

$$Z_{nm} = \frac{n+1}{\pi}\sum_x\sum_y f(x,y)V^*_{nm}(\rho,\theta), \quad x^2+y^2\le 1 \tag{5}$$

Suppose that one knows all moments Z_{nm} of $f(x,y)$ up to a given order $n_{\max}$. It is desired to reconstruct a discrete function $\hat{f}(x,y)$ whose moments exactly match those of $f(x,y)$ up to the given order $n_{\max}$. Zernike moments are the coefficients of the image expansion into orthogonal Zernike polynomials. By orthogonality of the Zernike basis

$$\hat{f}(x,y) = \sum_{n=0}^{n_{\max}}\sum_m Z_{nm}V_{nm}(\rho,\theta) \tag{6}$$

with m having similar constraints as in (1). Note that as $n_{\max}$ approaches infinity $\hat{f}(x,y)$ will approach $f(x,y)$.

The method of computing the Zernike moments by making use of (2), is called *direct method,* due to the fact that it computes the Zernike radial polynomials by using their definition formula.

2.2. Recursive computation

As it can be seen from Eq. (2), there are a lot of computations (factorials) that should be carried out, if the *direct method* chosen for the calculation of the Zernike moments. For this reason, many researchers have introduced fast computation methods [9,22–26] that eliminate the need of such time consumptive calculations. Among these, the well-known *q-recursive method* [9] is considered to be the most

efficient and is used in this paper. The method permits the evaluation of radial polynomials by using the following recursive equations

$$R_{nn}(r) = r^n \quad \text{for } n = m \tag{7}$$

$$R_{n(n-2)}(r) = nR_{nn}(r) - (n-1)R_{(n-2)(n-2)}(r) \quad \text{for } n-m = 2 \tag{8}$$

$$R_{n(m-4)} = H_1 R_{nm}(r) + \left(H_2 + \frac{H_3}{r^2}\right) R_{n(m-2)} \quad \text{otherwise} \tag{9}$$

where the coefficients H_1, H_2 and H_3 are given by:

$$H_1 = \frac{m(m-1)}{2} - mH_2 + \frac{H_3(n+m+2)(n-m)}{8}$$
$$H_2 = \frac{H_3(n+m)(n-m+2)}{4(m-1)} + (m-2) \tag{10}$$
$$H_3 = \frac{-4(m-2)(m-3)}{(n+m-2)(n-m+4)}.$$

When calculations of quantities through recurrence relations are involved in the main body of any algorithm, the danger of a numerical error, which may appear in a step of the algorithm, to be carried out to next steps, threads the stability and the accuracy of the whole procedure.

In the present paper, a detailed study of the way the computations of the *q-recursive method* are performed is attempted.

3. Numerical error analysis

It has already been discussed in Section 1 that there are some computational issues [7,8], regarding the accuracy with which the Zernike moments are computed. However, no research results have been reported on the numerical behaviour of the recursive equations, recently used for fast computation of the Zernike moments.

In this section, an exhaustive analysis of the numerical stability of the *q-recursive method* is introduced. There are two kinds of errors generated in the execution of this algorithm, the *overflow error*, due to computers' limitation about the maximum magnitude that a number can reach and the *finite precision error* due to the finite precision representation that a number can have.

3.1. Overflow error

Overflow is the situation in which a quantity takes a higher value, from the range of its data type. For example, the *float* (seven digit precision) data type has a valid range (1.18×10^{-45}, 3.40×10^{45}), while the *double* (15 digit precision) data type has a valid range (2.23×10^{-308}, 1.79×10^{308}), in the case of IBM PC compatible computers.

The appearance of an overflow condition is an undesirable event, for the accuracy of the algorithm, since it corrupts the final results. The study of the overflow error in the execution of the *q-recursive* algorithm follows.

$$R_{nn}(r) = r^n \tag{11}$$

This equation is the main source of an overflow error, since it includes raising to a power. Table 1 illustrates some combinations of the input parameters to the above equation, which generate an overflow. In order to clarify the presence of the overflow, two representations of the computed results, with 32 and 64 bit have been used.

$$R_{n(n-2)}(r) = nR_{nn}(r) - (n-1)R_{(n-2)(n-2)}(r) \tag{12}$$

The only case in which this equation generates an overflow is when the R_{nn} and $R_{(n-2)(n-2)}$ quantities have been computed by Eq. (11) with an overflow, too. Thus, Eq. (12) is responsible for the overflow propagation that might have been generated by Eq. (11). Table 2 depicts the way the overflow error generated by Eq. (11) influences Eq. (12).

$$R_{n(m-4)} = H_1 R_{nm}(r) + \left(H_2 + \frac{H_3}{r^2}\right) R_{n(m-2)} \tag{13}$$

Eq. (13) manifests behaviour similar to Eq. (12), in the sense that it propagates the overflow error generated in Eq. (11).

To conclude, the previous analysis shows that during the evaluation of the Zernike moments via the *q-recursive* algorithm, in certain cases some radial polynomials may be zeroed due to overflow. Therefore, according to Eq. (5), the presence of zeroed polynomials causes the disappearance of some quantities in the summation, and finally the inaccurate computation of the corresponding Zernike moment.

It is clear from the previous analysis that polynomial overflow depends on two factors, the magnitude of the polynomial degree n and the radius r of the corresponding pixel for which the polynomial is computed. Moreover, one can deduce the following results, in a straightforward manner:

Table 1

Input values			Output value
n	r		R_{nn}
23	0.01	64 bit	1.00000000×10^{-46}
		32 bit	$0.00000000\times10^{+00}$
24	0.01	64 bit	1.00000000×10^{-48}
		32 bit	$0.00000000\times10^{+00}$
25	0.01	64 bit	1.00000000×10^{-50}
		32 bit	$0.00000000\times10^{+00}$
26	0.01	64 bit	1.00000000×10^{-52}
		32 bit	$0.00000000\times10^{+00}$
27	0.01	64 bit	1.00000000×10^{-54}
		32 bit	$0.00000000\times10^{+00}$
27	0.02	64 bit	1.34217728×10^{-46}
		32 bit	$0.00000000\times10^{+00}$
28	0.01	64 bit	1.00000000×10^{-56}
		32 bit	$0.00000000\times10^{+00}$
28	0.02	64 bit	2.68435456×10^{-48}
		32 bit	$0.00000000\times10^{+00}$

Table 2

			Input values		Output value
n	r		R_{nn}	$R_{(n-2)(n-2)}$	$R_{n(n-2)}$
25	0.01	64 bit	1.00000000×10^{-50}	1.00000000×10^{-46}	$-2.39975000\times10^{-45}$
		32 bit	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$
26	0.01	64 bit	1.00000000×10^{-52}	1.00000000×10^{-48}	$-2.49974000\times10^{-47}$
		32 bit	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$
27	0.01	64 bit	1.00000000×10^{-54}	1.00000000×10^{-50}	$-2.59973000\times10^{-49}$
		32 bit	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$
28	0.01	64 bit	1.00000000×10^{-56}	1.00000000×10^{-52}	$-2.69972000\times10^{-51}$
		32 bit	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$
29	0.01	64 bit	1.00000000×10^{-58}	1.00000000×10^{-54}	$-2.79971000\times10^{-53}$
		32 bit	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$
29	0.02	64 bit	5.36870912×10^{-50}	1.34217728×10^{-46}	$-3.75653946\times10^{-45}$
		32 bit	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$
30	0.01	64 bit	1.00000000×10^{-60}	1.00000000×10^{-56}	$-2.89970000\times10^{-55}$
		32 bit	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$
30	0.02	64 bit	1.07374182×10^{-51}	2.68435456×10^{-48}	$-7.78140700\times10^{-47}$
		32 bit	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$	$0.00000000\times10^{+00}$

1. The closer r is to 1, the higher the value of the polynomial degree n, for which the overflow appears.
2. The closer r is to 0, the lower the value of the polynomial degree n, for which the overflow occurs.

   Therefore,
3. The computation of the Zernike moments for large images, presents more overflowed polynomials than that of small ones, since the same normalized coordinate system $[-1,1]$, is attached to further pixels and thus more pixels have low radius r.

3.2. Finite precision error

Although, the computational power of the modern computers is impressively increasing, the number representation always introduces flaws and errors in the computations.

These problems have to be taken under consideration, during the development of any algorithm, and especially the recursive ones. In fact, in the iterative algorithms a numerical error generated in one iteration may be accumulated or even amplified in subsequent recursion.

An efficient methodology, which explores the way the finite precision errors are generated and propagated, during a recursive algorithm, has been introduced and used in very popular signal processing algorithms [10–15]. This methodology employs a number of fundamental propositions demonstrating the way the four operations addition, multiplication, division and subtraction, influence the generation and transmission of the quantization error. These propositions are described in the following.

3.2.1. General remarks

The propositions stated in this paper hold true independently of the radix of the arithmetic system. However, the numerical error generation and propagation will be studied in the decimal representation, because the decimal arithmetic system is far more familiar and clear to users. In this arithmetic system, precision comparison between two numbers will be made in accordance with the following:

Definition 1. Consider two numbers, n_1 and n_2, written in the canonical exponential form, with the same number, n, of decimal digits in the mantissa, i.e.

$$n_1 = d_1 d_2 d_3 \cdots d_n \times 10^{\tau}, \quad n_2 = \delta_1 \delta_2 \delta_3 \cdots \delta_n \times 10^{\rho}$$

with

$$\tau \geq \rho$$

Then, these two numbers differ by K decimal digits, if and only if

$$||n_1| - |n_2|| = d \times 10^{\tau-(n-K)}, \quad 1 \leq d < 10$$

For example, according to this definition, the two numbers 1.234567 and 1.234912 differ indeed by three decimal digits, but the following two 1.000002 and 0.999996 differ by one decimal digit, as one might intuitively expect.

Definition 2. Let all quantities be written in the canonical exponential form, with n decimal digits in the mantissa. Suppose that the correct value of an arbitrary quantity α is α_c, if all calculations were made with infinite precision. Then, one may define that the quantity α has been computed with precisely the last λ decimal digits erroneous, if and only if, α and α_c differ λ digits according to Definition 1.

As will be evident from the subsequent analysis, all the formulas, that constitute a certain iterative algorithm, are not equivalent from the point of view of the finite precision error generation and propagation.

Notation. For any quantity a expressed in the canonical exponential form, we shall write: (i) man(a) for the mantissa of a, and (ii) $E(a)$ for the exponent of a.

Proposition 1. *Let all the involved quantities be computed with finite precision of n decimal digits in the mantissa, and consider any quantity computed by means of a formula of the type*

Multiplication $x = yz$

Suppose that, due to the previous finite precision calculations, the quantity y has been computed with precisely the last λ decimal digits erroneous, while z has been computed with up to λ decimal digits erroneous. Then,

(i) *if* $|\text{man}(y)\cdot\text{man}(z)| \geq 10$, *then x is computed with precisely the last λ or λ−1 decimal digits erroneous.*
(ii) *if* $|\text{man}(y)\cdot\text{man}(z)| < 10$, *then x is computed with precisely the last λ or λ+1 decimal digits erroneous.*

Proposition 2. *Let all the following quantities be computed with finite precision of n decimal digits in the mantissa, and consider any quantity computed through a formula of the type*:

Division $x = \dfrac{y}{z}$

Suppose that, due to the previous finite precision calculations, the quantity y has been computed with precisely the last λ decimal digits erroneous, while z has been computed with up to λ decimal digits erroneous. Then,

(iii) *if* $|\text{man}(y)/\text{man}(z)| \geq 1$, *then x is computed with precisely the last λ or λ−1 decimal digits erroneous.*
(iv) *if* $|\text{man}(y)/\text{man}(z)| < 1$, *then x is computed with precisely the last λ or λ+1 decimal digits erroneous.*

Proposition 3. *Let all the involved quantities be computed with finite precision of n decimal digits in the mantissa, and consider any quantity calculated through a formula of the type*

Subtraction $x = y - z$

with

$yz > 0$

where

$x = x_1x_2x_3\cdots x_n \times 10^{\delta}, \quad y = y_1y_2y_3\cdots y_n \times 10^{\tau},$

$z = z_1z_2z_3\cdots z_n \times 10^{\rho}$

are such that

$\delta < \max\{\tau,\rho\} \Leftrightarrow E(x) < \max\{E(y),E(z)\}.$

Let

$d = |\max\{\tau,\rho\} - \delta|$

Moreover, suppose that, due to the previous finite precision calculations, the higher order quantity say y has been computed with precisely the last λ decimal digits erroneous, while z has been computed with a number of erroneous decimal digits equal to, or smaller than λ. Then x is computed with the last (λ+d) decimal digits erroneous.

Proposition 4. (*The numerical error relaxation shift*). *Let all quantities be computed with finite precision of n decimal digits in the mantissa, and consider any quantity x computed through a sum of two quantities, that is*:

Addition $x = y + z$

Suppose, moreover, that z has its last λ decimal digits erroneous and that the exponent of z is by v smaller than the exponent of y, that is:

$v = E(y) - E(z) > 0.$

Then z transfers to x only λ−v erroneous decimal digits if λ−v >0 or it does not transfer finite precision error at all, if $\lambda - v \leq 0$.

3.2.2. Finite precision error in 'q-recursive' algorithm

In order to analyze the numerical behaviour of Eqs. (7)–(10), we proceed as follows [15]:

Step 1: We execute the algorithm with *n* digits precision for the mantissa
Step 2: In parallel, we execute the algorithm with 2*n* digits precision for the mantissa
Step 3: We cast any quantity z_{2n} computed by 2*n* decimal digits precision to a quantity $\bar{z}_n$ of *n* decimal digits precision.
Step 4: We compare quantities $\bar{z}_n$ and z_n, according to Definition 1 and in this way, we obtain the exact number of erroneous decimal digits with which each quantity z_n is computed.

$$R_{nn}(r) = r^n \tag{14}$$

The maximum finite precision error that is generated by this equation is up to eight erroneous decimal digits, due to the implementation of raising a number to a certain power. Thus, this equation generates a finite precision error owing to the *multiplication* operation. Table 3 shows examples of a significant finite precision error for some combinations of the inputs to the algorithm that computes Zernike moments. The erroneous decimal digits are marked with bold fonts in Table 3.

This type of finite precision error once generated in a step of this algorithm, it usually propagates in the subsequent iterations.

$$R_{n(n-2)}(r) = nR_{nn}(r) - (n-1)R_{(n-2)(n-2)}(r) \tag{15}$$

Eq. (15) introduces large finite precision error, due to the fact that the polynomials computed via Eq. (14) are quite frequently zeroed because of the overflow (see Table 4). Eq. (15) introduces another serious type of finite precision error, in the cases where the two quantities that participate in the subtraction operation, $nR_{nm}(r)$ and $(n-1)R_{(n-2)(n-2)}(r)$ both (i) are of opposite sign, and (ii) have a number of digits in common in the sense of Definition 1. This seems to occur quite frequently when the inputs to the algorithm that computes the Zernike moments are of the form described below.

In fact, if we write eq. (15), as

$$\begin{aligned} R_{n(n-2)}(r) &= nR_{nn}(r) - (n-1)R_{(n-2)(n-2)}(r) \\ &= nr^n - (n-1)r^{(n-2)} = r^n(n-(n-1)r^{-2}) \end{aligned} \tag{16}$$

Table 3

Input values			Output value	Finite precision error
n	r		$R_{n(n-2)}$	
26	0.02	64 bit	6.71088640×10^{-45}	Eight digits
		32 bit	$7.\mathbf{00649232}\times10^{-45}$	
29	0.03	64 bit	6.86303774×10^{-45}	Eight digits
		32 bit	$7.\mathbf{00649232}\times10^{-45}$	
32	0.04	64 bit	1.84467441×10^{-45}	Eight digits
		32 bit	$1.\mathbf{40129846}\times10^{-45}$	
34	0.05	64 bit	5.82076609×10^{-45}	Eight digits
		32 bit	$5.\mathbf{60519386}\times10^{-45}$	
41	0.08	64 bit	1.06338240×10^{-45}	Eight digits
		32 bit	$1.\mathbf{40129846}\times10^{-45}$	
43	0.09	64 bit	1.07752637×10^{-45}	Eight digits
		32 bit	$1.\mathbf{40129846}\times10^{-45}$	
45	0.10	64 bit	1.00000000×10^{-45}	Eight digits
		32 bit	$1.\mathbf{40129846}\times10^{-45}$	
46	0.11	64 bit	8.01795321×10^{-45}	Eight digits
		32 bit	$8.\mathbf{40779079}\times10^{-45}$	

then (i) and (ii), hold when:

$$R_{n(n-2)}(r) \cong 0 \tag{17}$$

Keeping in mind that Eq. (15), holds for $n \geq 2$ we can conclude that (i) and (ii) conditions are valid when the radius r has the following values with respect to the Zernike moment of order n:

$$r \cong \sqrt{\frac{n-1}{n}} \tag{18}$$

From the above, the following result could be stated, also justified by Table 5.

Result 1. Consider all pixels having distance r from the centre of the image, such that the radial polynomials of order n, $(n-2)$ are close to zero, namely $R_{n(n-2)}(r) \cong 0$. Then, due to the subtraction operation in Eq. (15), a significant amount of finite precision error is immediately generated, making the results of the q-recursive algorithm totally unreliable soon. In particular, the Zernike moments of order n and any repetition m, are fully inaccurate:

$$R_{n(m-4)} = H_1 R_{nm}(r) + \left(H_2 + \frac{H_3}{r^2}\right) R_{n(m-2)} \tag{19}$$

3.2.2.1. Quantity H_3. The maximum finite precision error with which this quantity is computed is equal to two erroneous decimal digits, where this error is caused due to the *division* operation. This error appears for many combinations of (n,m) see Table 6, no matter what value the radius r, takes.

3.2.2.2. Quantity H_2. The maximum finite precision error with which this quantity is computed is equal to three erroneous decimal digits. Responsible for this error is the quantity

$$\frac{H_3(n+m)(n-m+2)}{4(m-1)}$$

Table 4

			Input values		Output value	Finite precision error
n	r		R_{nn}	$R_{(n-2)(n-2)}$	$R_{n(n-2)}$	
34	0.04	64 bit	2.95147905×10^{-48}	1.84467441×10^{-45}	$-6.07739052\times10^{-44}$	Nine digits
		32 bit	$0.00000000\times10^{+00}$	1.40129846×10^{-45}	$-4.62428493\times10^{-44}$	
41	0.07	64 bit	4.45676403×10^{-48}	9.09543680×10^{-46}	$-3.61990199\times10^{-44}$	Nine digits
		32 bit	$0.00000000\times10^{+00}$	1.40129846×10^{-45}	$-5.60519386\times10^{-44}$	
43	0.08	64 bit	6.80564734×10^{-48}	1.06338240×10^{-45}	$-4.43694178\times10^{-44}$	Nine digits
		32 bit	$0.00000000\times10^{+00}$	1.40129846×10^{-45}	$-5.88545355\times10^{-44}$	
45	0.09	64 bit	8.72796357×10^{-48}	1.07752637×10^{-45}	$-4.70184018\times10^{-44}$	Nine digits
		32 bit	$0.00000000\times10^{+00}$	1.40129846×10^{-45}	$-6.16571324\times10^{-44}$	
47	0.10	64 bit	1.00000000×10^{-47}	1.00000000×10^{-45}	$-4.55300000\times10^{-44}$	Nine digits
		32 bit	$0.00000000\times10^{+00}$	1.40129846×10^{-45}	$-6.44597294\times10^{-44}$	
49	0.11	64 bit	1.06718957×10^{-47}	8.81974853×10^{-46}	$-4.18118700\times10^{-44}$	Nine digits
		32 bit	$0.00000000\times10^{+00}$	1.40129846×10^{-45}	$-6.72623263\times10^{-44}$	
51	0.12	64 bit	1.09205258×10^{-47}	7.58369846×10^{-46}	$-3.73615455\times10^{-44}$	Nine digits
		32 bit	$0.00000000\times10^{+00}$	1.40129846×10^{-45}	$-7.00649232\times10^{-44}$	
77	0.25	64 bit	4.37905770×10^{-47}	7.00649232×10^{-46}	$-4.98774672\times10^{-44}$	Nine digits
		32 bit	$0.00000000\times10^{+00}$	1.40129846×10^{-45}	$-1.06498683\times10^{-43}$	

Table 5

Input values			Output value	Finite precision error
n	r		$R_{n(n-2)}$	
2	$\sqrt{\frac{1}{2}}$	64 bit	2.22044605×10^{-16}	17 digits
		32 bit	$-5.96046448\times10^{-08}$	
6	$\sqrt{\frac{5}{6}}$	64 bit	4.44089210×10^{-16}	18 digits
		32 bit	2.38418579×10^{-07}	
13	$\sqrt{\frac{12}{13}}$	64 bit	8.88178420×10^{-16}	18 digits
		32 bit	4.76837158×10^{-07}	
15	$\sqrt{\frac{14}{15}}$	64 bit	$-1.77635684\times10^{-15}$	17 digits
		32 bit	9.53674316×10^{-07}	
18	$\sqrt{\frac{17}{18}}$	64 bit	$-3.55271368\times10^{-15}$	17 digits
		32 bit	$-9.53674316\times10^{-07}$	
19	$\sqrt{\frac{18}{19}}$	64 bit	$-1.77635684\times10^{-15}$	17 digits
		32 bit	9.53674316×10^{-07}	

due to the *division* operation. This error appears for many combinations of (n,m) see Table 7, independently of the value of r.

3.2.2.3. Quantity H_1. The maximum finite precision error with which this quantity is computed is equal to six erroneous decimal digits. Responsible for this error is quantity

$$\frac{m(m-1)}{2}-mH_2$$

due to the *subtraction* operation and according to the results of Proposition 3.

The above quantities introduce a limited number of finite precision errors, since they are constant and they are not computed recursively. Thus, they affect the computation of Eq. (9) slightly, without creating a serious instability to the algorithm execution (Table 8).

3.2.2.4. Quantity (H_2+H_3/r^2). This quantity introduces a significant finite precision error, for the same reason described in connection with Eq. (15). As it can be seen from Table 9, there is a set of (n,m,r) for which conditions of Proposition 3 hold. In this case quantity H_2+H_3/r^2, takes an unreliable value. Conditions of Proposition 3, are satisfied when

$$r\cong\sqrt{\frac{4(m-1)(m-3)}{(3-m)(n+m)(n-m+2)+(n+m-2)(n-m+4)(m-1)}} \tag{20}$$

Finally, the obtained $R_{n(m-2)}$ polynomial, presents a finite precision error of nine erroneous decimal digits (Tables 10 and 11). For the values of radius r that the experiments have been performed (from $r=0$ to 1 with step of 0.01), it seems that the high finite precision error of the H_2+H_3/r^2 quantity has been relaxed due to the *summation* operation (Proposition 4).

However, the algorithm still offers totally unreliable results when r satisfies (20), due to the large finite precision error generated during the execution of the formula H_2+H_3/r^2.

Table 6

Input values			Output value	Finite precision error
n	m		H_3	
4	4	64 bit	$-3.33333333\times10^{-01}$	Two digits
		32 bit	$-3.33333343\times10^{-01}$	
8	6	64 bit	$-6.66666667\times10^{-01}$	Two digits
		32 bit	$-6.66666687\times10^{-01}$	
10	4	64 bit	$-6.66666701\times10^{-02}$	Two digits
		32 bit	$-6.66666667\times10^{-02}$	
11	11	64 bit	$-3.60000000\times10^{+00}$	Two digits
		32 bit	$-3.59999990\times10^{+00}$	
12	6	64 bit	$-3.00000000\times10^{-01}$	Two digits
		32 bit	$-3.00000012\times10^{-01}$	
12	8	64 bit	$-8.33333333\times10^{-01}$	Two digits
		32 bit	$-8.33333313\times10^{-01}$	
13	13	64 bit	$-4.58333333\times10^{+00}$	Two digits
		32 bit	$-4.58333349\times10^{+00}$	
14	4	64 bit	$-3.57142857\times10^{-02}$	Two digits
		32 bit	$-3.57142873\times10^{-02}$	

Table 7

Input values			Output value	Finite precision error
n	m		H_2	
24	14	64 bit	$2.81318681\times10^{+00}$	Three digits
		32 bit	$2.81318569\times10^{+00}$	
26	24	64 bit	$8.05072464\times10^{+00}$	Three digits
		32 bit	$8.05072594\times10^{+00}$	
30	24	64 bit	$5.31237458\times10^{+00}$	Three digits
		32 bit	$5.31237602\times10^{+00}$	
34	24	64 bit	$4.16770186\times10^{+00}$	Three digits
		32 bit	$4.16769981\times10^{+00}$	
34	28	64 bit	$6.09876543\times10^{+00}$	Three digits
		32 bit	$6.09876442\times10^{+00}$	
34	30	64 bit	$7.81757508\times10^{+00}$	Three digits
		32 bit	$7.81757736\times10^{+00}$	
35	15	64 bit	$2.36011905\times10^{+00}$	Three digits
		32 bit	$2.36011791\times10^{+00}$	
35	27	64 bit	$5.12820513\times10^{+00}$	Three digits
		32 bit	$5.12820625\times10^{+00}$	

Table 8

Input values			Output value	Finite precision error
n	m		H_1	
37	37	64 bit	$-4.85339506\times10^{-01}$	Six digits
		32 bit	$-4.85229492\times10^{-01}$	
44	42	64 bit	$-6.49245064\text{e}\times10^{-01}$	Six digits
		32 bit	$-6.49139404\times10^{-01}$	
47	47	64 bit	$-4.88657845\times10^{-01}$	Six digits
		32 bit	$-4.88525391\times10^{-01}$	
48	48	64 bit	$-4.88909009\times10^{-01}$	Six digits
		32 bit	$-4.89013672\times10^{-01}$	
50	42	64 bit	$-8.10298103\times10^{-01}$	Six digits
		32 bit	$-8.10180664\times10^{-01}$	
53	39	64 bit	$-8.60818713\times10^{-01}$	Six digits
		32 bit	$-8.60717773\times10^{-01}$	
53	43	64 bit	$-8.33695180\times10^{-01}$	Six digits
		32 bit	$-8.33862305\times10^{-01}$	
54	40	64 bit	$-8.61637557\times10^{-01}$	Six digits
		32 bit	$-8.61511230\times10^{-01}$	

Table 9

				Input values		Output value	Finite precision error
n	m	r		H_2	H_3/r^2	H_2+H_3/r^2	
20	10	0.5	64 bit	$2.28571429\times10^{+00}$	$-2.28571429\times10^{+00}$	8.88178420×10^{-16}	18 digits
			32 bit	$2.28571415\times10^{+00}$	$-2.28571439\times10^{+00}$	$-2.38418579\times10^{-07}$	
43	21	0.56	64 bit	$2.70620347\times10^{+00}$	$-2.70610726\times10^{+00}$	9.62171469×10^{-05}	Seven digits
			32 bit	$2.70620346\times10^{+00}$	$-2.70610738\times10^{+00}$	9.60826874×10^{-05}	
84	46	0.65	64 bit	$3.33201058\times10^{+00}$	$-3.33192449\times10^{+00}$	8.60962399×10^{-05}	Eight digits
			32 bit	$3.33200836\times10^{+00}$	$-3.33192468\times10^{+00}$	8.36849213×10^{-05}	
85	51	0.7	64 bit	$3.77065200\times10^{+00}$	$-3.77062058\times10^{+00}$	3.14218382×10^{-05}	Seven digits
			32 bit	$3.77065277\times10^{+00}$	$-3.77062058\times10^{+00}$	3.21865082×10^{-05}	
92	34	0.46	64 bit	$2.43988270\times10^{+00}$	$-2.43917312\times10^{+00}$	7.09577635×10^{-04}	Seven digits
			32 bit	$2.43988419\times10^{+00}$	$-2.43917298\times10^{+00}$	7.11202621×10^{-04}	
92	54	0.69	64 bit	$3.68313866\times10^{+00}$	$-3.68403330\times10^{+00}$	$-8.94635390\times10^{-04}$	Seven digits
			32 bit	$3.68313599\times10^{+00}$	$-3.68403339\times10^{+00}$	$-8.97407532\times10^{-04}$	
92	56	0.71	64 bit	$3.88826899\times10^{+00}$	$-3.88866093\times10^{+00}$	$-3.91934333\times10^{-04}$	Seven digits
			32 bit	$3.88826752\times10^{+00}$	$-3.88866115\times10^{+00}$	$-3.93629074\times10^{-04}$	
97	47	0.59	64 bit	$2.96693203\times10^{+00}$	$-2.96714694\times10^{+00}$	$-2.14912916\times10^{-04}$	Seven digits
			32 bit	$2.96693039\times10^{+00}$	$-2.96714711\times10^{+00}$	$-2.16722488\times10^{-04}$	

Table 10

Input values							
n	m	r		H_1	R_{nm}	H_2+H_3/r^2	$R_{n(m-2)}$
22	8	0.92	64 bit	$-7.45129870\times10^{-01}$	2.25499251×10^{-01}	$1.59930216\times10^{+00}$	1.05059722×10^{-01}
			32 bit	$-7.45128632\times10^{-01}$	2.25495577×10^{-01}	$1.59930229\times10^{+00}$	1.05059162×10^{-01}
24	4	0.01	64 bit	$-6.85714286\times10^{-01}$	1.28505642×10^{-12}	$-1.99811429\times10^{+03}$	$-4.99471275\times10^{-09}$
			32 bit	$-6.85712814\times10^{-01}$	1.26065575×10^{-12}	$-1.99811450\times10^{+03}$	$-4.89987384\times10^{-09}$
24	2	0.01	64 bit	$-5.84415584\times10^{-01}$	$-4.99471275\times10^{-09}$	$-7.77558442\times10^{+02}$	9.98000601×10^{-06}
			32 bit	$-5.84416389\times10^{-01}$	$-4.89987384\times10^{-09}$	$-7.77558472\times10^{+02}$	9.79050765×10^{-06}
24	0	0.01	64 bit	$-3.29059829\times10^{-01}$	9.98000601×10^{-06}	$-1.26863248\times10^{+02}$	$-7.76003500\times10^{-03}$
			32 bit	$-3.29059780\times10^{-01}$	9.79050765×10^{-06}	$-1.26863258\times10^{+02}$	$-7.61268940\times10^{-03}$
28	24	0.02	64 bit	$-4.80109739\times10^{-01}$	2.68435456×10^{48}	$-3.00790754\times10^{+04}$	$-1.81118771\times10^{-43}$
			32 bit	$-4.80102539\times10^{-01}$	$0.00000000\times10^{+00}$	$-3.00790781\times10^{+04}$	$-1.89175293\times10^{-43}$
28	26	0.02	64 bit	$-6.36923077\times10^{-01}$	$-1.81118771\times10^{-43}$	$-1.76835938\times10^{+04}$	5.44788517×10^{-39}
			32 bit	$-6.36917114\times10^{-01}$	$-1.89175293\times10^{-43}$	$-1.76835957\times10^{+04}$	5.69021904×10^{-39}
28	22	0.02	64 bit	$-7.53968254\times10^{-01}$	$-9.63381886\times10^{-35}$	$-7.91174603\times10^{+03}$	1.11209604×10^{-30}
			32 bit	$-7.53952026\times10^{-01}$	$-1.00623533\times10^{-34}$	$-7.91174707\times10^{+03}$	1.16156474×10^{-30}
28	20	0.02	64 bit	$-7.78032037\times10^{-01}$	1.11209604×10^{-30}	$-5.53948284\times10^{+03}$	$-8.79862139\times10^{-27}$
			32 bit	$-7.78045654\times10^{-01}$	1.16156474×10^{-30}	$-5.53948291\times10^{+03}$	-9.19000614×10^{27}

3.3. Discussion

The main purpose in this paper was the detailed study of the generation and propagation of numerical errors, during the recursive computation of Zernike moments, by using the 'q-recursive' method. The main observation of this work is not the occurrence of overflow errors, which other researchers may have already notified, but the generation and propagation of finite precision errors. Regarding overflow errors, our study relates them not only to the moment order but to the image size as well. It is actually concluded that the larger the image size is the smaller the moment order presenting overflow error is. This fact is mentioned in the resulting statements at the end of Section 3.1. Regarding finite precision error, two conditions were defined; Eqs. (18) and (20), according to which a finite precision error may be generated and propagated, no matter what the moment order is. Table 5 demonstrates the existence of finite precision errors (due to condition (18)) even in low orders ($n=2$, 6, 13, 15, 18, 19), which are essential for both reconstruction and classification purposes. Finite precision error due to condition (20) also occurs in medium to high orders, however according to Proposition 4 this error, although present in intermediate calculations, is relaxed by the summation operation. The reason why these finite precision errors are generated is the existence of 'ill-posed' subtractions according to Proposition 3. While the finite precision errors occurr even in small moment orders, the previous study includes simulations with high moment orders. A question about the usefulness of the higher order moments in image processing and pattern classification, can be stated as, 'up to which moment order has one to compute in order to reconstruct and recognize an image, with minimum errors?'. For reconstruction purposes our experience shows that moment orders up to 30 are useful when the image size is 256×256 and above. The authors have presented such studies in reference [6]. For pattern classification purposes, the number of required moments is in general small and includes mostly low orders. However, depending on the problem at hand, there might be necessary to include some moments of higher order in order to improve the discriminative power of the moment feature

Table 11

Input values				Output value	Finite precision error
n	m	r		$R_{n(m-4)}$	
22	12	0.92	64 bit	$-3.98715399\times10^{-06}$	Nine digits
			32 bit	$-1.86264515\times10^{-06}$	
24	8	0.01	64 bit	9.98000601×10^{-06}	Eight digits
			32 bit	9.79050765×10^{-06}	
24	6	0.01	64 bit	$-7.76003500\times10^{-03}$	Eight digits
			32 bit	$-7.61268940\times10^{-03}$	
24	4	0.01	64 bit	9.84459960×10^{-01}	Eight digits
			32 bit	9.65767384×10^{-01}	
28	28	0.02	64 bit	5.44788517×10^{-39}	Eight digits
			32 bit	5.69021904×10^{-39}	
28	26	0.02	64 bit	$-9.63381886\times10^{-35}$	Eight digits
			32 bit	$-1.00623533\times10^{-34}$	
28	22	0.02	64 bit	$-8.79862139\times10^{-27}$	Eight digits
			32 bit	$-9.19000614\times10^{-27}$	
28	20	0.02	64 bit	4.87398113×10^{-23}	Eight digits
			32 bit	5.09078827×10^{-23}	

vector. This was shown in [19], in which the appropriate Zernike moment set was selected using a simple genetic algorithm. In that paper, it was concluded that a more suitable moment set can be derived, according to the problem being processed, in terms of its recognition rate. This moment set is not restricted to low order moments but it may also include other higher moments as well. Finally, we have to stress here that the approach presented in this paper does not stand in simple observations regarding the appearance of finite precision error in 'q-recursive' algorithm but it gives a thorough analysis on the conditions governing the numerical behaviour of the algorithm.

4. Conclusions

An exact analysis of the finite precision error generation and propagation for the *q-recursive* algorithm that computes the radial polynomials needed for the computation of the Zernike moments is presented in this paper. Two kinds of numerical error have pointed the *overflow error* and the *finite precision error*. The error due to overflow is generated in Eq. (11) of the algorithm and propagates to the subsequent iterations by means of formulae (12) and (13). The overflow error is directly connected to the order of the polynomial n and the order of the pixel's radius r. The second type of error, the finite precision error, is caused by the nature of the fundamental operations involved in the recurrent equations of the algorithm. It is demonstrated that there are sets of (n,m,r) variables that make the values of the radial polynomials totally unreliable. The exact conditions quantities (n,m,r) must satisfy in order to destroy the results of the *q-recursive* algorithm, are stated. The study introduced in this paper constitutes a first attempt to analyse the numerical instability of the fast recursive algorithms employed for the computation of the popular Zernike moments. The results of the paper stress the necessity of developing more numerically stable, relevant recursive algorithms. Moreover, exhaustive comparative studies of the finite precision error generation and propagation in the various recursive algorithms computing the Zernike moments must take place. In this way, a proper trade off between computational speed and numerical stability must be determined for these algorithms. An attempt to tackle the aforementioned problems will be undertaken in future work.

References

[1] M.K. Hu, Visual pattern recognition by moment invariants, IRE Trans. Inform. Theory IT-8 (1962) 179–187.

[2] M. Teague, Image analysis via the general theory of moments, J. Opt. Soc. Am. 70 (8) (1980) 920–930.

[3] A. Khotanzad, J.-H. Lu, Classification of invariant image representations using a neural network, IEEE Trans. Acoust. Speech Signal Processing 38 (6) (1990) 1028–1038.

[4] A. Khotanzad, Y.H. Hong, Invariant image recognition by zernike moments, IEEE Trans. Pattern Anal. Machine Intell. PAMI-12 (5) (1990) 489–497.

[5] G.A. Papakostas, D.A. Karras, B.G. Mertzios, Image coding using a wavelet based Zernike moments compression technique, 14th International Conference on Digital Signal Processing (DSP2002), vol. II, Santorini-Hellas (Greece), 1–3 July, 2002, pp. 517–520.

[6] G.A. Papakostas, D.A. Karras, B.G. Mertzios, Y.S. Boutalis, An efficient feature extraction methodology for computer vision applications using wavelet compressed zernike moments, ICGST International Journal on Graphics, Vision and Image Processing, Special Issue: Wavelets and Their Applications SI1 (2005) 5–15.

[7] R. Mukundan, S.H. Ong, P.A. Lee, Discrete vs. continuous orthogonal moments for image analysis, in: Proceedings of the International Conference on Imaging Science Systems and Technology, vol. 1, 2001, pp. 23–29.

[8] S.X. Liao, M. Pawlak, On the accuracy of zernike moments for image analysis, IEEE Trans. Pattern Anal. Machine Intell. 20 (12) (1998) 1358–1364.

[9] C.W. Chong, P. Raveendran, R. Mukundan, A comparative analysis of algorithms for fast computation of zernike moments, Pattern Recognition 36 (2003) 731–742.

[10] C.N. Papaodysseus, E.B. Koukoutsis, C.N. Triantafyllou, Error sources and error propagation in the levinson-durbin algorithm, IEEE Trans. Signal Process. 41 (4) (1993).

[11] C.N. Papaodysseus, G. Carayannis, E.B. Koukoutsis, E. Kayafas, Comparing LS FIR filtering and l-Step ahead linear prediction, IEEE Trans. Signal Process. 41 (2) (1993).

[12] C. Papaodysseus, E. Koukoutsis, C. Vassilatos, Error propagation and methods of error correction in LS FIR, IEEE Trans. Signal Process. 42 (5) (1994).

[13] C. Papaodysseus, E. Koukoutsis, G. Stavrakakis, C.C. Halkias, Exact analysis of the finite precision error generation and propagation in the FAEST and the fast transversal algorithms: a general methodology for developing robust RLS schemes, Math. Comput. Simul. 44 (1997) 29–41.

[14] Y. Boutalis, C. Papaodyseus, E. Koukoutsis, A new multichannel recursive least squares algorithm for very robust and efficient adaptive filtering, J. Algorithms 37 (2000) 283–308.

[15] C. Papaodysseus, C. Alexiou, T.H. Panagopoulos, G. Rousopoulos, D. Kravaritis, A novel general methodology for studying and remedying finite precision error with application in kalman filtering, Stoch. Environ. Res. Ris. Assess. 17 (2003) 1–19.

[16] R. Mukundan, Some computational aspects of discrete orthonormal moments, IEEE Trans. Image Process. 13 (8) (2004) 1055–1059.

[17] M. Zhenjiang, Zernike moment-based image shape analysis and its application, Pattern Recognit. Lett. 21 (2000) 169–177.

[18] C. Kan, M.D. Srinath, Invariant character recognition with zernike and orthogonal fourie-mellin moments, Pattern Recognit. 35 (2002) 143–154.

[19] G.A. Papakostas, Y.S. Boutalis, B.G. Mertzios, Evolutionary selection of Zernike moment sets in image processing, 10th International Workshop on Systems, Signals and Image Processing (IWSSIP'03), Prague–Czech Republic, 10–11 September, 2003.

[20] T.W. Lin, Y.F. Chou, A comparative study of Zernike moments for image retrieval, Proceedings of 16th IPPR Conference on Computer Vision, Graphics Image Process. (CVGIP 2003), 2003, pp. 621–629.

[21] D.G. Sim, H.K. Kim, R.H. Park, Invariant texture retrieval using modified zernike moments, Image Vis. Comput. 22 (2004) 331–342.

[22] R. Mukundan, A contour integration method for the computation of Zernike moments of a binary image, Proceedings of National Conference on Research and Development in Computer Science and its Applications, Penang–Malaysia, 27–29 November, 1997, pp. 188–192.

[23] J. Gu, H.Z. Shu, C. Toumoulin, L.M. Luo, A novel algorithm for fast computation of zernike moments, Pattern Recognit. 35 (2002) 2905–2911.

[24] C.W. Chong, P. Raveendran, F. Takeda, New computational methods for full and subset zernike moments, Inform. Sci. 159 (2004) 203–220.

[25] R. Mukundan, K.R. Ramakrishnan, Fast computation of legendre and zernike moments, Pattern Recognit. 28 (9) (1995) 1433–1442.

[26] S.O. Belkasim, M. Ahmadi, M. Shridhar, Efficient algorithm for fast computation of zernike moments, J. Franklin Inst. 333(B) (4) (1996) 577–581.

[27] F. Zernike, Beugungstheorie des schneidenverfahrens und seiner verbesserten form, der phasenkonstrastmethode, Physica 1 (1934) 689–701.