Package 'IM'

February 19, 2015

Type Package

Title Orthogonal Moment Analysis

Version 1.0

Date 2012-07-18

Author Bartek Rajwa, Murat Dundar, Allison Irvine, Tan Dang

Maintainer Allison Irvine <a.irvine2@gmail.com>

Contact Tan Dang <dangt@purdue.edu>, Bartek Rajwa <rajwa@cyto.purdue.edu>

Description Compute moments of images and perform reconstruction from moments.

License GPL (>= 2)

LazyLoad yes

Depends R (>= 2.10.0),png (>= 0.1-4),jpeg (>= 0.1-2),bmp (>= 0.1), methods

Collate ImageClass.R MethodDefinitions.R ComplexImageClass.R OrthogonalImageClass.R MultipleImagesClass.R OrthMoments.R MultiCmplxWrapper.R gpzmWrapper.R frWrapper.R fmWrapper.R fcWrapper.R

NeedsCompilation yes

Repository CRAN

Date/Publication 2013-03-06 18:46:11

R topics documented:

`IM-package` *Image Moment (IM) - Package*

Description

Compute Generalized-Pseudo Zernike, Fourier-Chebyshev, Fourier-Mellin, Radial Harmonic Fourier, Dual Hahn, Discrete Chebyshev, Krawtchouk, Gegenbauer, Legendre, Chebyshev moment type of single and multiple images.

Legend: gpzm, fc, fm, fr, hahn, cheby, krawt, gegen, legend, chebycont.

Details

Package:	IM
Type:	Package
Version:	1.0
Date:	2012-07-18
License:	GPL (>= 2)
LazyLoad:	yes
Depends:	R (>= 2.9.0),png (>= 0.1-4),jpeg (>= 0.1-2),bmp (>= 0.1)

Class `linkS4class{Image}` parent class for CmplxIm and OrthIm classes
Class `CmplxIm` handles moment analysis of single image with type : Generalized Pseudo-Zernike, Fourier-Chebyshev, Fourier-Mellin, Radial Harmonic Fourier.
Class `OrthIm` handles moment analysis of single image with type: Dual Hahn, Discrete Chebyshev, Krawtchouk, Gegenbauer, Legendre, Chebyshev.
Class `MultiIm` handles moments analysis of list of matrices with type: Generalized Pseudo-Zernike, Fourier-Chebyshev, Fourier-Mellin, Radial Harmonic Fourier, Dual Hahn, Discrete Chebyshev, Krawtchouk, Gegenbauer, Legendre, Chebyshev.

Function `momentObj` provides easy construction of object of different moment type and parameter.

Parameter requirements:

Generalized Pseudo-Zernike:	a is natural number. When a=0, the polynomials are equivalent to Pseudo-Zernike
Dual Hahn:	a, c are integers, with a >= 0 and a > abs(c)-1
Krawtchouk:	a is a real number between 0 and 1
Gegenbauer:	a is real number and a > 1/2

See references for more details.

Author(s)

Allison Irvine, Tan Dang.

References

Hosny, K.M.(2011). Image representation using accurate orthogonal Gegenbauer moments. *Pattern Recognition Letters*, 32(6), 795–804.

Mukundan, R., Ong, S.H., and Lee, P.A.(2001). Image analysis by Tchebichef moments. *Image Processing, IEEE Transactions on*, 10(9), 1357–1364.

Ping, Z., Wu, R., and Sheng, Y. (2002). Image description with Chebyshev-Fourier moments. *Journal of the Optical Society of America A*, 19(9), 1748–1754.

Ren, H., Liu, A., Zou, J., Bai, D., and Ping, Z.(2007). Character Reconstruction with Radial-Harmonic-Fourier Moments. *Fuzzy Systems and Knowledge Discovery*, 2007. FSKD 2007.

Sheng, Y., and Shen, L. (2007). Orthogonal Fourier-Mellin moments for invariant pattern recognition. *Journal of the Optical Society of America A*, 11(6), 1748–1757.

Xia, T., Zhu, H., Shu, H., Haigron, P., and Luo, L. (2007). Image description with generalized pseudo-Zernike moments. *Journal of the Optical Society of America A*, 24(1), 50–59.

Venkataramana, A., and Raj, P.A.(2007). Image Watermarking Using Krawtchouk Moments. *Computing: Theory and Applications, 2007. ICCTA '07. International Conference on*, 5-7 March 2007, 676–680.

Zhu, H.(2012). Image representation using separable two-dimensional continuous and discrete orthogonal moments. *Pattern Recognition*, 45(4), 1540–1558.

Zhu, H., Shu, H., Zhou, J., Luo, L. and Coatrieux, J.L.(2007) Image Analysis by Discrete Orthogonal Dual Hahn Moments. *Pattern Recognition Letters*, 28(13), 1866–1704.

`calcCentroid` *Calculate the centroid of a grayscale image*

Description

Calculate the centroid of a grayscale image using geometric moments. The centroid is based on pixel intensity. The centroid, located at pixel coordinates (x0,y0) is calculated using the following equation:

$$x0 = \frac{m_{10}}{m_{00}}$$

$$y0 = \frac{m_{10}}{m_{00}}$$

where

$$m_{pq} = \sum_{1}^{N} \sum_{1}^{M} (x^p)(y^q) f(x, y)$$

N is the horizontal dimensionality of the image and M is the vertical dimensionality of the image.

$$f(x, y)$$

is the pixel intensity of the pixel at location (x,y) in the image.

Usage

```
calcCentroid(I)
```

Arguments

`I`	An image represented as a 2-dimensional grayscale matrix.

Value

numeric: `(x0, y0)`

`x0`	x-coordinate of centroid
`y0`	y-coordinate of centroid

Author(s)

Allison Irvine, Tan Dang

References

Flusser, J., Suk, T., Zitova, B. Moments and Moment Invariants in Pattern Recognition. 2009. John Wiley & Sons Ltd. ISBN: 978-0-470-69987-4

See Also

Image,CmplxIm,OrthIm,polarXY

Examples

```
data(pirate)
I=rowSums(img,dims=2);
center= calcCentroid(I);
```

`calcMaxRadius` *Compute maximum radius.*

Description

Compute maximum distance of all pixels from the centroid of the image. Alternatively, it is the radius of the the smallest disk that covers the entire image with its origin at the centroid. The radius is determined by:

$$r = \frac{m_{00}}{2}\sqrt{\frac{N}{M} + \frac{M}{N}}$$

where

$$m_{pq} = \sum_{1}^{N}\sum_{1}^{M}(x^p)(y^q)f(x,y)$$

Usage

```
calcMaxRadius(I, center)
```

Arguments

`I`	A 2-dimensional image matrix.
`center`	The x and y coordinates of the image centroid, input as `c(x0, y0)`. If missing, the centroid will be calculated automatically.

Value

`maxRadius`	The radius of the the smallest disk that covers the entire image with its origin at the centroid.

Note

The value calculated by this function is used to obtain the polar coordinates of the pixels in the image. The coordinates of the pixels are scaled with this value so that they will be between -1 and 1 before the polar coordinates are calculated. This is necessary because complex moments are only defined on the unit disk.

Author(s)

Tan Dang

See Also

CmplxIm,calcCentroid,polarXY

Examples

```
data(mandril)
I=rowSums(img,dims=2);

maxRadius= calcMaxRadius(I);
maxRadius= calcMaxRadius(I, calcCentroid(I));
```

---

checkOrder *Find the maximum order of finite moments*

Description

Function to find the maximum order (and repetition for complex moments) of moments such that all moment values up to that order (in both the x and y directions) are finite and not Nan values. The maximum order will be at most the maximum order represented in the matrix of moment values. This function looks for the largest size rectangular block in the moment matrix such that no values are infinity or NaN.

Usage

```
checkOrder(moments)
```

Arguments

moments	Matrix containing image moments from order 0 up to some order p in the x-direction and q in the y-direction, or in the case of complex moments, up to order p and repetition q.

Value

orderP	The maximum order along the x-axis of the moments matrix, which is also the maximum number of valid columns minus 1.
orderQ	The maximum order along the y-axis of the moments matrix, which is also the maximum number of valid rows minus 1.

Note

The maximum order of p and q returned will be the same in the current implementation, representing the upper bounds of a square block within the moments matrix.

Author(s)

Allison Irvine

See Also

OrthIm

Examples

```
data(lena);

obj= momentObj(img, "cheby", c(300,200));

maxOrder= checkOrder(obj@moments);
```

CmplxIm-class *CmplxIm Class*

Description

This class contains an image and all information about the image required for computing the moments of the image. The object has methods for computing moments, moment invariants, and reconstructing the image from moments.
The types of moments that can be calculated using this object are: generalized Pseudo-Zernike, Fourier Mellin, Fourier Chebyshev, and Radial Harmonic Fourier.

Details

For generalized Pseudo-Zernike ("gpzm") moments, a single parameter is required,

$$a$$

. A larger value of

$$a$$

decreases the range of the polynomial values used for moment computation.

Objects from the Class

Objects can be created by calls of the form `new("CmplxIm", img, filename)`. First the constructor checks for an input argument containing an image (img). This can be a multidimensional numeric array or a matrix. The image is converted to grayscale by summing over all colors and stored in slot "I". If an input image is not provided, a filename (filename) can be passed to the constructor. The constructor will then read in the image file and convert it to grayscale. If a filename is given, it must have the extension ".jpg", ".png", or ".bmp". If "img" and "filename" are not provided, and empty object will be created.

Slots

`radius`: Object of class `"matrix"` Contains the radius of all of the pixels with respect to the centroid of the image.

`theta`: Object of class `"matrix"` Contains the angle of all of the pixels with respect to the centroid.

`constants`: Object of class `"matrix"` Contains constants used in the calculation of generalized pseudo-zernike moments and image reconstruction.

`params`: Object of class `"numeric"` Contains the parameters used to calculate generalized pseudo-zernike moments.

`order`: Object of class `"numeric"` Contains the order and repetition used to calculate moments. Two values may be given, which specify order and repetition, except in the case of generalized Pseudo-Zernike moments. In this case only one value should be provided, specifying order. Then repetition is constrained by the order.

`I`: Object of class `"matrix"` Contains the original image, in grayscale.

`dimensions`: Object of class `"numeric"` Contains the dimensions of the image.

`centroid`: Object of class `"numeric"` Contains the x and y coordinates of the centroid (center point) of the image, used for calculating continuous orthogonal moments.

`filename`: Object of class `"character"` Contains the filename of the image.

`imType`: Object of class `"character"` Contains the original image file type.

`momentType`: Object of class `"character"` Contains the type of moments being calculated. Choices are: "gpzm" (Generalized Pseudo-Zernike),"fm" (Fourier Mellin),"fc" (Fourier Chebyshev),"fr" (Harmonic Radial Fourier).

`moments`: Object of class `"matrix"` Contains the moments calculated from the image. Dimensions are order by repetition.

`invariant`: Object of class `"matrix"` Contains the invariants computed for the image. Dimensions are order in y direction by order in x direction.

`reconstruction`: Object of class `"matrix"` Contains the image reconstructed from the moments.

`error`: Object of class `"character"` containing a list of error messages produced from calling class methods.

Extends

Class `"Image"`, directly.

Methods

Moments<- `signature(obj = "CmplxIm")`: Calculate the moments of the image. Before calling this method the momentType, params, and order must be set. Usage: Moments(obj) <- 0

Invariant<- `signature(obj = "CmplxIm")`: Calculate moment invariants of the image by taking the absolute values of the moments and normalizing them by the moment with order and repetition 0. Before calling this method the momentType, params, and order must be set. Usage: Invariant(obj) <- 0

Reconstruct<- `signature(obj = "CmplxIm")`: Reconstruct the image from the moments, using moments up to a specified order. If no order value is given, the image will be reconstructed from all available moments. Useage: Reconstruct(obj) <- order

initialize `signature(.Object = "CmplxIm")`: Constructor. A call to new("CmplxIm",img,filename) calls this function and creates an CmplxIm object. The object slots I, centroid, radius, theta, and dimensions are set. (And filename and imType if a filename was provided)

momentType<- `signature(obj = "CmplxIm")`: Set the moment type for the moments to be calculated.

setImage<- `signature(obj = "CmplxIm")`: Set "I", the image in the object. Input can be a matrix or multidimensional numeric array. When this is called, the object slots centroid, radius, theta, and dimensions are set.

setOrder<- `signature(obj = "CmplxIm")`: Set the order up to which to calculate moments in the x and y directions.

setParams<- `signature(obj = "CmplxIm")`: Set the parameters to be used for calculating moments.

setPolar<- `signature(obj = "CmplxIm")`: Calculate the polar coordinates of the pixels with respect to the centroid. The object slots "radius" and "theta" are set.

Author(s)

Allison Irvine

See Also

calcCentroid, polarXY, plotMoment, displayImg, checkOrder, polarTransform, Image, OrthIm

Examples

```
#load image
data(earth)
#initialize object
obj<- new("CmplxIm", img=img)
## Not run: displayImg(obj@I)
#set the moment type to generalized Pseudo-Zernike
momentType(obj)<- "gpzm"
#set the order
setOrder(obj)<- 25
#set the parameter
setParams(obj)<- 1

## Not run:
#calculate moments of the image
Moments(obj)<- NULL
#calculate rotation invariants
Invariant(obj) =NULL;
#reconstruct the image from moments
Reconstruct(obj)<- c(20,20)

## End(Not run)
## Not run: plotMoment(obj);
```

```
## Not run: displayImg(obj@reconstruction);
```

---

datasets *Image Sample Dataset*

Description

Sample images provided with this package are lena, pirate, mandril, livingroom, earth, and earth_200. Two sets of images used to demonstrate classification are provided. One dataset, "bacteria", contains light scatter images of 4 different types of bacteria. There are 5 images from each class. When the data is loaded, the variable "img" contains the images, and "labels" contains the type of bacteria of each image in the image list. The "characters" dataset contains images of 8 different handwritten Tamil characters. There are 40 images for each class.

Usage

```
data(lena)
data(pirate)
data(mandril)
data(livingroom)
data(earth)
data(earth_200)
data(bacteria)
data(characters)
```

Format

RBG image

Source

http://www.imageprocessingplace.com/root_files_V3/image_databases.htm
http://en.wikipedia.org/wiki/Earth
Bartek Rajwa, Bindley Bioscience Center, Discovery Park, Purdue University
http://www.hpl.hp.com/india/research/penhw-interfaces-1linguistics.html

References

Image Processing Place. Retrieved on 07/18/2012 from http://www.imageprocessingplace.com/root_files_V3/image_databas
HP Labs India. Retrived on 08/13/2012 from http://www.hpl.hp.com/india/research/penhw-interfaces-1linguistics.html

demoPoly *Calculate and display polynomials used to calculate image moments*

Description

Displays a plot of polynomials of a given order or plots of multiple orders. The polynomials are plotted over a range of pixel coordinates. Discrete and continuous orthogonal polynomials are plotted for a given order over a range of a single dimension, and complex polynomials are plotted over 2 dimensions.

Usage

```
demoPoly(order, type, N, params)
```

Arguments

order	The order of polynomials to be displayed. A single order or array of orders must be specified.
type	The type of polynomial which will be displayed.
N	The maximum dimension up to which polynomials will be calculated. The polynomials displayed will be plotted from 0 to N along the x-axis. This argument represents the dimensionality of the image for which moments would be calculated.
params	The parameters of the polynomials to be calculated. The parameters required depend on the polynomial type.

Details

For complex polynomials, (generalized Pseudo-Zernike,Fourier Chebyshev,Radial Harmonic Fourier, and Fourier Mellin), both order and repetition must be specified for each plot, for example as: `(order, repetition)`. To plot multiple orders and repetition for all complex polynomials, the input argument would be in the form of a 2-dimensional matrix where the first column is order and the second column is the corresponding repetition.
For real polynomials, (Chebyshev, Dual Hahn, Krawtchouk, Gegenbauer, Legendre, continuous Chebyshev), each polynomials will be plotted over a single dimension of pixel coordinates for a single set of parameters and a specified range of orders. All of the polynomials in this case will be plotted in the same graph.

Value

polynomials: the displayed polynomials will be returned as a list, with each element in the list being a polynomial of a specific order. This value is only returned for real polynomials, (Chebyshev, Dual Hahn, Krawtchouk, Gegenbauer, Legendre, continuous Chebyshev).

Author(s)

Allison Irvine, Tan Dang

See Also

OrthIm, CmplxIm, MultiIm

Examples

```
#display chebyshev polynomials of orders 20 to 25 for dimensionality 256
demoPoly(order = 20:25,type = "cheby",N = 256)

## Not run:
#display generalized Pseudo-Zernike polynomials of order and repetition (3,2), (4,2), (5,3), (6,3)
#for pixel dimensions of 300x300, with paramater a=0
demoPoly(rbind(c(3,2), c(4,2), c(5,3), c(6,3)), "gpzm", 300, 0);

## End(Not run)

## Not run:
#display Radial Fourier polynomials of order and repetition (3,2), (4,2), (5,3), (6,3)
#for pixel dimensions of 300x300,
demoPoly(rbind(c(3,2), c(4,2), c(5,3), c(6,3)), "fr", 300);

## End(Not run)
```

---

displayImg *Display an image in grayscale*

Description

Display an image in grayscale. If the image is in color, it will still be displayed in grayscale. All images displayed will be processed with histogram equalization.

Usage

```
displayImg(img)
```

Arguments

img	A matrix or numeric array representation of the image. If the image is not in grayscale, it will still be displayed in grayscale. If using a CmplxIm, Image, or OrthIm object, the original image would be represented as `obj@I` and the reconstructed image would be `obj@reconstruction`. For MultiIm object, function takes in list of images. In this case the input argument would be `obj@imageList` to access the original images or `obj@reconstruction`to access the reconstructed images. If the input is a list of images, they will all be plotted in one window.

Author(s)

Allison Irvine, Tan Dang

See Also

Image,CmplxIm,OrthIm, MultiIm

Examples

```
#### display the original image in a "OrthIm" object.
data(pirate);

#create OrthIm object from image
obj <- new("OrthIm",img = img,filename = "");
momentType(obj) = "cheby";
setOrder(obj) = c(200,200);

#display original image
displayImg(obj@I);

## Not run:
#analyze image
Moments(obj) = 0;
Reconstruct(obj) = NULL;

#display reconstructed image
displayImg(obj@reconstruction)

## End(Not run)


## Not run:
#### display a list of images from a "MultiIm" object.
data(characters)
#take a small subset
img = img[1:5];
#analyze images
obj= momentObj(I=img, type = "cheby", order = c(dim(img[[1]])[2],dim(img[[1]])[1]) );
Reconstruct(obj)= NULL;

#display original images
displayImg(obj@imageList);

#display reconstructed images
displayImg(obj@reconstruction);

## End(Not run)
```

`histeq` *Perform histogram equalization on an image.*

Description

Perform histogram equalization on an image.

Usage

```
histeq(I)
```

Arguments

I	A matrix or numeric array representation of an image.

Value

An image of equal dimensions to the input, with histogram equalization applied to all color channels.

Author(s)

Tan Dang

See Also

displayImg

Examples

```
data(pirate);
#perform histogram equalization
img = histeq(img);
#convert to grayscale
img = rowSums(img, dims=2)/3;
#rotate image 270 degrees so it appears upright
img = rotate270(img);
#set colors
levels = seq(0,1,.0000001);
g = gray(levels);
## Not run: image(img,col=g)
```

Image-class *Basic class to handle image moment analysis*

Description

This class contains general information about an image, and is inherited by the "CmplxIm" and "OrthIm" classes. It contains basic utilities and object slots which are needed for both continuous complex moment analysis and discrete/continuous real orthogonal moment analysis. An object of this class can be cast to either a CmplxIm type object or a OrthIm type object.

Creating objects

Objects will be created by calls of the form `new("Image", img, filename)`. When the constructor is called and a filename is given, the input file will be loaded and the image will be converted to grayscale and stored in slot "I". Only files of type "jpeg", "bmp" or "png" may be imported this way. If the argument "img", an image in matrix or numeric form, is provided when the constructor is called, the image will be converted to grayscale and stored in slot "I" of the object. When this slot value is set, the centroid and dimensions of the image will be calculated. The constructor for this class will be called when objects of the inheriting classes `"CmplxIm"` and `"OrthIm"` are created. See `OrthIm`, `CmplxIm`.

Slots

`I`: Object of class `"matrix"` representing the image.

`dimensions`: Object of class `"numeric"` containing dimension of the image.

`centroid`: Object of class `"numeric"` containing centroid of the image. See `calcCentroid`.

`filename`: Object of class `"character"` containing name of the image

`imType`: Object of class `"character"` containing image extension.

`momentType`: Object of class `"character"` containing moment type. Either `"gpzm"`, `"fm"`, `"fr"`, `"fc"`, `"krawt"`, `"hahr`
See `OrthIm`, `CmplxIm`.

`moments`: Object of class `"matrix"` containing computed moments from the image.

`invariant`: Object of class `"matrix"` containing computed invariants from the image.

`reconstruction`: Object of class `"matrix"` representing the reconstructed image from the computed moment.

`error`: Object of class `"character"` containing a list of error messages produced from calling class methods.

Methods

initialize `signature(.Object = "Image")`: Construct the `"Image"` object. To be used internally when constructing `"OrthIm"` and `"CmplxIm"` object only.

plotMoment `signature(obj = "Image")`: Plot the calculated moments. Usage: plotMoment(obj).

setCentroid<- `signature(obj = "Image")`: Set centroid of image to an array of 2 values (the x and y pixel coordinates of the centroid). If a value is not supplied, the centroid will be calculated using geometric moments. Usage: setCentroid(obj)<- c(value1,value2).

setImage<- `signature(obj = "Image")`: Set the image slot of the object to an image in matrix or numeric form. Usage: setImage(obj)<- img.

Note

This class does not have functions available for calculating moments or reconstruction from moments. In order to calculate moments you must use an object of type "OrthIm" or "CmplxIm".

Author(s)

Allison Irvine

See Also

OrthIm, CmplxIm, calcCentroid, plotMoment

Examples

```
# create an object of class "Image"
data(lena)
obj = new("Image",img=img)
#display image
## Not run: displayImg(obj@I)
#convert the object to the class "OrthIm"
obj=as(obj,"OrthIm")
```

---

momentObj *Calculate moments of image*

Description

Computes continuous complex or discrete/continuous orthogonal moments for an image or list of images.

Usage

```
momentObj(I, type, order, paramsX, paramsY)
```

Arguments

I	Either an image in matrix representation or a list of images.
type	One string or an array of two strings indicate the moment types to be calculated. Two types can be given only if they are both either continuous or discrete orthogonal moment types (not complex types). In this case, the moments in the x direction will be of first type and the moments in the y direction will be of the second type.
order	In the case of real orthogonal moments, one or two numbers may be provided to indicate the orders up to which to calculate moments in the x and y directions of the image. If one value is provided, the order will be the same in the x and y directions. In the case of complex moments (except for generalized Pseudo-Zernike), two values should be provided to specify the order and repetition. If generalized Pseudo-Zernike moments are being calculated, repetition is constrained by order, so only one value should be specified to indicate order.
paramsX	Parameters required for calculation of moments in the x direction. These are only required for certain types of moments: generalized Pseudo-Zernike, Hahn, Krawtchouk, and Gegenbauer. If the same moment types and parameters are to be used to calculate moments in both the x and y directions, the parameters may be specified only by this argument.
paramsY	Parameters required for calculation of moments in the y direction. These are only required for certain types of moments: generalized Pseudo-Zernike, Hahn, Krawtchouk, and Gegenbauer.

Details

For generalized Pseudo-Zernike ("gpzm") moments, a single parameter is required,

$$a$$

. A larger value of

$$a$$

decreases the range of the polynomial values used for moment computation.
For Gegenbauer moments, a single parameter is required,

$$a$$

with the constraints

$$a > 1$$

and

$$a\tilde{} = 0$$

.A larger value of

$$a$$

increases the values of the polynomials used for moment calculation. If

$$a = 1$$

, the moments are equivalent to continuous Chebyshev moments.
For Hahn moments, two parameters are required,

$$a$$

and

$$c$$

with the constraints

$$a > -\frac{1}{2}$$

and

$$a > abs(c) - 1$$

.

$$a$$

specifies where the polynomials are centered and the difference between

$$a$$

and

$$c$$

is positively correlated to the range of the polynomial values used to calculate moments.
For Krawtchouk moments, a single parameter is required,

$$a$$

with the constraint

$$0 < a < 1$$

.

$$a$$

specifies where the polynomials are centered, relative to the image centroid.
See references for further details on these parameters.
See the documentation for OrthIm,CmplxIm,and MultiIm for details about image reconstruction.

Value

If only one image is provided to the function in the first argument "I", an object of type "CmplxIm" or "OrthIm" will be returned, depending on the moment type specified.
If the argument "type", specifying the moment type to be calculated, is "gpzm","fm","fc",or "fr", an object of type "CmplxIm" will be returned.
If the argument "type", specifying the moment type to be calculated, is "cheby","chebycont","hahn","krawtchouk","gegen", or "legend", an object of type "OrthIm" will be returned.
If a list of images is provided to the function in the first argument "I", an object of type "MultiIm" will be returned.

Author(s)

Allison Irvine, Tan Dang

See Also

OrthIm,CmplxIm,MultiIm

Examples

```
#compute chebyshev moments of an image
data(lena)
## Not run: displayImg(obj@I)
obj = momentObj(I=img,type="cheby",order=c(500,500));
## Not run: plotMoment(obj)
#reconstruct the image from the moments
Reconstruct(obj) = c(200,200);
## Not run: displayImg(obj@reconstruction)


## Not run:
#calculate bivariate Legendre/Gegenbauer moments up to orders 50 and 100 with parameter 1 for Gegenbauer moments.
data(lena);
displayImg(obj@I);
obj= momentObj(img,c("legend", "gegen"),c(200, 300),NULL,2);
plotMoment(obj);
#reconstruct the image from all the moments
Reconstruct(obj) = NULL;
displayImg(obj@reconstruction);

## End(Not run)
```

```
## Not run:
#compute Radial Harmonic-Fourier moment invariants of a list of images, using up to order 10 and repetition 10
data(bacteria);
obj= momentObj(I=img[1:10],type="fr",order=c(10,10));
Invariant(obj) = NULL;

## End(Not run)
```

`MultiIm-class` *MultiIm Class*

Description

The class contains a list of images and all information required for computing the moments of the images. The images must have the same dimensions and centroid. The object has methods for computing moments and reconstructing the image from moments. Types of moments that can be calculated for this class are: generalized pseudo-zernike, fourier mellin, fourier chebyshev, radial fourier, krawtchouk, hahn, continuous chebyshev, legendre, and gegenbauer.

Creating objects

Objects can be created by calls of the form `new("MultiIm", images)` where `"images"` is list of matrices. The dimensions of the images will be checked for consistency when the constructor is called. Moment type, order, and parameters can be set using the class methods.

Slots

`imageList`: Object of class `"list"` Contains a list of image matrices of the same dimension.

`dimension`: Object of class `"numeric"` Contains the dimensions of the images.

`polynomials`: Object of class `"list"` Contains the polynomial values used for computing moments.

`storePoly`: Object of class `"logi"` Default is FALSE. Set to `"TRUE"` to store polynomial values. For gpzm, fm, fr, and fc moment types, if object@storePoly== TRUE, polynomial values will be stored in the slot "polynomials". Polynomial values for other moment types will always be stored. Set directly by object@storePoly<- TRUE

`momentType`: Object of class `"character"` Contains the moment type.

`order`: Object of class `"numeric"` Contains the maximum order up to which to calculate moments.

`Params`: Object of class `"list"` Contains the parameters used to calculate the moments. First list element is the parameters for the moments over x coordinates, second is for the y coordinates.

`moments`: Object of class `"list"` Contains a list of the moments calculated for each image.

`invariant`: Object of class `"list"` Contains a list of the invariants calculated for each image.

`reconstruction`: Object of class `"list"` Contains the list of reconstructed images.

`error`: Object of class `"character"` Contains the last error message produced from calling a class method.

Methods

momentType<- `signature(obj = "MultiIm")`: Set the moment type for the moments to be calculated. Takes one character string or an array of 2 character strings, for example c("type1","type2"). Usage: momentType(obj)<- c("krawt", "hahn") or momentType(obj)<-"gpzm"

setOrder<- `signature(obj = "MultiIm")`: Set the orders up to which to calculate moments in the x and y directions in the case of real orthogonal moments, or specify order and repetition for complex moment types. Usage: setOrder(obj)<- c(50,50)

setParams<- `signature(obj = "MultiIm")`: Set the parameters to be used for calculating moments. If two types of moments are being used, this should be a list of the parameters to be used for each moment type being used. Usage: setParams(obj)<- list(c(0.5), c(1,1))

transform<- `signature(obj = "MultiIm")`: Perform a transform of the images which creates an image of the pixels rearranged by polar coordinates. The rows are radius from the centroid and columns are angle. See `polarTransform`.

Moments<- `signature(obj = "MultiIm")`: Calculate the moments of a list of images. Before calling this method the momentType, params, and order must be set. Usage: Moments(obj)<- 0.

Invariant<- `signature(obj = "MultiIm")`: Calculate the moment invariants w.r.t. rotation of a list of images. Before calling this method the momentType, params, and order must be set. The method assumes that all images have the same centroid location. For complex moments, the magnitude of the moments are invariant to rotation. Usage: Invariant(obj)<- NULL.
For real orthogonal moments (continuous or discrete), the method normalizes all images prior to computing moments using the same procedure as `Invariant<-,OrthIm-method`. Usage: Invariant(obj)<- c(5,120)

plotPoly `signature(obj = "MultiIm", order = "list")`: Plot the polynomials used to calculate moments of a specified order or array of orders. Usage: plotPoly(obj, list(3:6, 1:4)).

Reconstruct<- `signature(obj = "MultiIm")`: Reconstruct the images from moments. Any order lower than or equal to the maximum order calculated can be used for reconstruction. Usage: Reconstruct(obj)<- c(50,50)

Author(s)

Tan Dang

See Also

`plotPoly` `displayImg` `Image`, `OrthIm` `CmplxIm`,

Examples

```
# Load sample data
data(bacteria);

#create MultiIm object, calculate generalized Pseudo-Zernike moments, invariants, and reconstruction for all imag
obj <- new("MultiIm", img);
momentType(obj)= c("gpzm");
setOrder(obj)= 20;
```

```
setParams(obj)= 1; # <--- GPZM has 1 parameter
obj@storePoly= TRUE;

## Not run:
Moments(obj)= NULL;
Invariant(obj)= NULL;
Reconstruct(obj)= NULL;
#display a subset of polynomials stored in the object
plotPoly(obj, order=rbind(c(3,2), c(4,2), c(5,3), c(6,3)));
displayImg(obj@reconstruction[1:5]);

## End(Not run)

## Not run:
#create MultiIm object, calculate discrete Chebyshev moments, invariants, and reconstruction for all images
obj <- new("MultiIm", img);
momentType(obj)= "cheby"
setOrder(obj)= 100;
Moments(obj)= 0;
Reconstruct(obj)= 75;
Invariant(obj)= c(7,100); # resolution is 7, scale is 100
displayImg(obj@reconstruction[1:4]);

## End(Not run)

## Not run:
#create MultiIm object, calculate continuous Chebyshev - Legendre moments, invariants, and reconstruction for all
obj <- new("MultiIm", img);
momentType(obj)= c("chebycont", "legend");
setOrder(obj)= c(100, 110);
transform(obj)= 8;  # perform polar transform on images
Moments(obj)= 0;
Reconstruct(obj)= NULL;

# order of pairs (20,10), (21, 11),...(28,18) will be plotted
plotPoly(obj, order=cbind(20:28, 10:18));
displayImg(obj@reconstruction[1:6]);

## End(Not run)
```

`OrthIm-class` *Class for handling continuous or discrete orthogonal moments analysis of an image*

Description

This class contains an image and all information about the image required for computing a moments analysis of the image. The object can be passed to functions to compute moments and reconstruct the image from the moments. The types of moments that can be calculated using this object are: chebyshev, krawtchouk, hahn, continuous chebyshev, legendre, and gegenbauer.

Creating Objects

Objects can be created by calls of the form `new("OrthIm", img, filename)`. First the constructor checks for an input image (img). This can be a multidimensional numeric array or a matrix. The image is converted to grayscale by summing over all colors and stored in slot "I". If an input image is not provided, a filename (filename) can be passed to the constructor. The constructor will then read in the image file and convert it to grayscale. If a filename is given, it must have the extension ".jpg", ".png", or ".bmp".

Slots

`polynomials`: Object of class `"list"` Contains the polynomials used to calculate moments. First list element contains the polynomials over the x coordinates, the second contains polynomials over the y coordinates. The number of rows is the pixel coordinates, the number of columns is the order of the polynomials.

`params`: Object of class `"list"` Contains the parameters used to calculate the moments. First list element is the parameters for the moments over x coordinates, second is for the y coordinates.

`order`: Object of class `"numeric"` Contains the order used to calculate moments. Two values may be given to calculate different orders in the x and y directions.

`I`: Object of class `"matrix"` Contains the original image, in grayscale.

`dimensions`: Object of class `"numeric"` Contains the dimensions of the image.

`centroid`: Object of class `"numeric"` Contains the x and y coordinates of the centroid (center point) of the image, used for calculating continuous orthogonal moments.

`filename`: Object of class `"character"` Contains the filename of the image.

`imType`: Object of class `"character"` Contains the original image file type.

`momentType`: Object of class `"character"` Contains the types of moments being calculated. Up to two different types may be selected, the first being the type of moments calculated in the x direction, the second being for the y direction. If two different types are chosen, they must both be either discrete or continuous. Choices of discrete moments are: "cheby","krawt","hahn". Choices of continuous moments are: "chebycont","gegen","legendre".

`moments`: Object of class `"matrix"` Contains the moments calculated from the image. Dimensions are order in y direction by order in x direction.

`reconstruction`: Object of class `"matrix"` Contains the image reconstructed from the moments.

`error`: Object of class `"character"` Contains the last error message produced from calling a class method.

Extends

Class `"Image"`, directly.

Methods

Moments<- `signature(obj = "OrthIm")`: Calculate the moments of the image. Before calling this method the momentType, params, and order must be set. Usage: Moments(obj) <- 0

Invariant<- `signature(obj = "OrthIm")`: Calculate the invariants of the image by normalization before computing moments. `polarTransform` will be called with `resolution=value[1], scale=value[2]`. Before calling this method the momentType, params, and order must be set. Usage: Invariant(obj) <- c(6,100)

Reconstruct<- `signature(obj = "OrthIm")`: Reconstruct the image from the moments, using moments up to a specified order. If no order value is given, the image will be reconstructed from all available moments. Useage: Reconstruct(obj) <- order

initialize `signature(.Object = "OrthIm")`: Constructor. A call to new("OrthIm",img,filename) calls this function and creates an OrthIm object. The object slots I, centroid and dimensions are set. (And filename and imType if a filename was provided)

momentType<- `signature(obj = "OrthIm")`: Set the moment type for the moments to be calculated. Takes one character string or an array of 2 character strings, for example c("type1","type2")

plotPoly `signature(obj = "OrthIm", order = "numeric")`: Plot the polynomials used to calculate the moments in this object. The order, can be 1 number or several numbers, and the polynomials of those orders will be plotted for all x coordinates.

setOrder<- `signature(obj = "OrthIm")`: Set the order up to which to calculate moments in the x and y directions.

setParams<- `signature(obj = "OrthIm")`: Set the parameters to be used for calculating moments.

transform<- `signature(obj = "OrthIm")`: Perform a transform of the image which creates an image of the pixels rearranged by polar coordinates. The rows are radius from the centroid and columns are angle. The method calls `polarTransform` with `padded=FALSE, scale=0`

Author(s)

Allison Irvine

See Also

`OrthMoments`, `OrthReconstruct`, `calcCentroid`, `plotMoment`, `displayImg`, `checkOrder`, `polarTransform`, `Image`, `CmplxIm`

Examples

```
data(mandril);
obj <- new("OrthIm",img = img,filename = "");
momentType(obj) = c("chebycont","legend");
setOrder(obj) = c(150,150);
Moments(obj) = NULL;
Reconstruct(obj) = c(125,125);
## Not run:
displayImg(list(obj@I,obj@moments,obj@reconstruction));
plotPoly(obj,order=c(100,150));

## End(Not run)
```

plotMoment *Display a heat map of image moments*

Description

Displays image moments as a heat map. A inverse hyperboic sine transform is used to scale the values. If the moments are complex numbers, the absolute value is plotted.

Usage

```
plotMoment(obj)
```

Arguments

obj	A Image, OrthIm, CmplxIm, MultiIm class object

Author(s)

Allison Irvine

See Also

Image, CmplxIm, OrthIm, MultiIm, momentObj

Examples

```
data(livingroom)
Obj= momentObj(img, type="gpzm", order=20, 0);
plotMoment(Obj);

## Not run:
images= list(img,img,img,img);
Obj= momentObj(images, type="krawt", order=100, 0.5);
plotMoment(Obj);

## End(Not run)
```

polarTransform *Represent an image by plotting radius against angle*

Description

Perform a polar transform of an image by calculating the polar coordinates of the pixels in an image and rearranging the pixels into a plot of angle (theta) against radius. The polar coordinates are calculated with respect to the centroid, determined by `calcCentroid` or the center of the image space. All transformed images will be shifted to principal axis determined by:

$$\theta = \frac{1}{2} arctan(\frac{2\mu_{00}}{\mu_{10} - \mu_{01}})$$

where

$$\mu_{pq} = \sum_{1}^{N} \sum_{1}^{M} (x - x0)^p (y - x0)^q f(x, y)$$

with (x0, y0) are centroid computed by `calcCentroid`

Usage

```
polarTransform(I, resolution, scale, center)
```

Arguments

`I`	Image represented as a matrix or numeric array
`resolution`	An integer which determines the number of angle values between 0 and 2pi to be represented in the transform.
`scale`	Integer. Default is 0, no scaling in the transformation. If set to a positive integer, the maximum radius used in the transformation will be set to `scale`, and the transformed image will have dimensions `scale, (scale*resolution)`
`center`	The x and y coordinates of the centroid to be used in the transform. If this argument is not provided, the center of the image space will be the centroid.

Details

This transformation is performed by finding the Euclidean coordinates of points which are contained within the smallest circle with origin at the centroid or center of image which can be contained within the image boundaries. The pixels whose coordinates most closely match these Euclidean coordinates are plotted at the corresponding polar coordinates. At each radius, the perimeter is divided into `radius * resolution` equally spaced point. Then, the Euclidean coordinates of the points will be used in the transformation. This results in a upper-triangular matrix. This transform was created to approximate rotational invariance for orthogonal moments. `revPolar` is the inverse transform of `polarTransform`.

Value

`PI`	The transformed image.
`pAxis`	The principal axis of the image.
`resolution`	The resolution used for the transformation.
`scale`	The scale used for the transformation.
`center`	The centroid used for the transformation.

Author(s)

Allison Irvine, Tan Dang

See Also

revPolar, polarXY, calcCentroid

Examples

```
data(circles);
I=rowSums(img,dims=2)
R=polarTransform(I, 6, 100);
displayImg(list(I,R[[1]]));
```

---

polarXY *Calculate polar coordinates of elements in a matrix*

Description

Computes the radius and and angle of each element's location in a matrix with respect to the centroid.

Usage

```
polarXY(I, centroid)
```

Arguments

I	"Matrix" representation of an image, or any matrix.
centroid	two numbers, the x coordinate of the centroid and the y coordinate of the centroid. The function calcCentroid may be used to estimate the image centroid.

Value

"list"

radius	The radius of the polar coordinates of each element in the input matrix.
theta	The angle of the polar coordinates of each element in the input matrix.

Author(s)

Allison Irvine, Tan Dang

See Also

polarTransform, calcCentroid

Examples

```
data(lena);
I=rowSums(img,dims=2);
center= calcCentroid(I);
result= polarXY(I, center);
```

`revPolar` *Inverse transform of the* `polarTransform` *method*

Description

Performs an inverse polar transform of an image by calculating the Euclidean coordinates of the rearranged pixels in an image transformed by the `polarTransform` method. See the documentation for polarTransform for details on the calculation of the polar image coordinates.

Usage

```
revPolar(d,params)
```

Arguments

`d`	The dimensions of the original image in Euclidean space.
`params`	A list containing the resulting transformed image and parameters from the polarTransform method. This list contains the following items: PI: The transformed image returned by the function `polarTransform`. pAxis: The principal axis used to shift the polar coordinates of the pixels in the transformed image. resolution: An integer which determines the number of angle values between 0 and 2pi which was used to create the transform. center: The x and y coordinates of the centroid used in the transform. scale: The scaling factor used to transform the image.

Details

This function is only for recovering the original image from an image transformed using the polarTransform method. The input arguments to `revPolar`, `PI`,`pAxis`,`resolution`,`center`,`scale` are returned by the `polarTransform` method in a list. Note that the dimensions of the original image, the first argument `d`, must also be specified.

Value

`IR`	The inverse transformed image; an approximation of the original image from the polar transformed image.

Author(s)

Allison Irvine, Tan Dang

See Also

polarTransform, polarXY, calcCentroid

Examples

```
#perform a polar transform on the original image
data(circles);
I=rowSums(img,dims=2)
R=polarTransform(I, 20, 100);
## Not run: displayImg(R[[1]]);

#now reverse the transform
IR = revPolar(dim(I),R);
## Not run: displayImg(IR);
```

---

rotate270 *Rotate an image (or matrix) 270 degrees.*

Description

Rotate an image (or matrix) 270 degrees.

Usage

```
rotate270(img)
```

Arguments

img	A matrix or numeric array representation of an image.

Value

An image of equal dimensionality to the input, rotated 270 degrees.

Author(s)

Allison Irvine

See Also

displayImg

Examples

```
data(pirate);
#perform histogram equalization
img = histeq(img);
#convert to grayscale
img = rowSums(img, dims=2)/3;
#rotate image 270 degrees so it appears upright
img = rotate270(img);
#set colors
levels = seq(0,1,.0000001);
g = gray(levels);
image(img,col=g)
```

Index